www.clubedelphi.net Feita para desenvolvedores

ClubeDelphi

Ano II - Edição 20 - R$6,90

Boletos e Cartão de Crédito

Saiba como gerar boletos bancários e realizar cobrança por cartão de crédito em suas aplicações **Delphi Win32**.

IBX/FB
Exibindo dicas na forma de balão com a descrição sua foi substituída no campo de busca

FTP
Implementando atualização automática de sua aplicação, via internet

Usando Ado.Net Expert

ClientDataSet - Filtros Avançados

IBX/FB - IBCompare
Comparar bancos de dados e gerar scripts automáticos para a correção de diferenças

ASP.NET - Mão na Massa
Criando um menu recursivo para navegação de categorias e produtos

MTS - Delphi 2006
Configurações para formar o IDE mais rápido

NÃO PERCA

PORTAL DO ASSINANTE

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:

- Mais de 220 Vídeo Aulas!
- Diversos Mini-Cursos Gratuitos!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 140 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse www.devmedia.com.br/clubedelphi/portal.asp e utilize as informações abaixo:

Login: DVM.PL

Senha: A55LY

O acesso é válido por 30 dias a partida da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

Corpo Editorial

Diretor Editorial e Administrativo
Gladstone Matos
gladstone@neoficio.com.br

Designer
Vinicius O. Andrade
viniciusoandrade@gmail.com

Capa
Antonio Xavier
jaxs_design@yahoo.com.br

Articulistas desta edição
Rodrigo Sendin, Marcos Alexandre Miguel, Marco Antônio Pereira Araújo, Fabricio Desbessel, Adriano Santos, Murilo Costa Monteiro Filho, Jorge Luis Bublitz, Rafaela de Campos, Rodrigo Otto Mostaert, Danilo Rocha Valente e Fernando Nomellini

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Luciano Pimenta
lucianopimenta@clubedelphi.net

Revisão
Fabio Correa
fabiocorrea@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Publicidade
Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:
Kaline Dolabella
publicidade@devmedia.com.br

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Thiago Andrade – Atendimento ao Leitor
www.devmedia.com.br/central/default.asp
(21) 2220-5375

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 2220-5375

Parcerias

Para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:
Jeff Wendell
jeff@devmedia.com.br

ASSESPRO PARANÁ

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão! Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

Luciano Pimenta - Editor do Site
lucianopimenta@clubedelphi.net

ClubeDelphi

Ano 6 - 79ª Edição - 2006 - ISSN 1517990-7 **Impresso no Brasil**

Editorial

Acredito que o termo VoIP não seja novidade para mais ninguém. VoIP é uma tecnologia que permite a digitalização, codificação da voz e o encapsulamento desse dado em pacotes IP, para transmissão em uma rede que utilize o protocolo TCP/IP. Através dela, aplicações podem por exemplo codificar sua voz em forma de bits e trafegar pela Internet. Softwares como o Skype, MSN Messenger e Google Talk são bons exemplos de uso de VoIP. Permitem que pessoas conversem pela internet, usando seu próprio computador como se fosse uma aparelho telefônico, e o melhor de tudo, na maioria das vezes sem pagar um centavo. E que tal implementar seu próprio software de VoIP usando Delphi? É o que mostra o Jorge, em um artigo exclusivo para esta edição. Uma curiosidade: o próprio Skype (www.skype.com) foi feito em Delphi e é hoje uma das principais aplicações na área.

Ainda nesta edição, confira duas matérias sobre backups no IB/FB. No artigo da Rafaela, conheça a ferramenta FIBS (Firebird-Interbase Backup Scheduler), aplicativo Open Source voltado para backup de bancos Firebird e InterBase. Permite que o desenvolvedor ou DBA agende os backups a serem realizados, defina local de destino, mirrors, frequência (diariamente, semanalmente etc.). Indispensável para quem trabalha como IB/FB. Em outro artigo, o Rodrigo mostra como você pode construir sua própria ferramenta de backup utilizando os componentes nativos do Delphi, como a classe TIBBackupService. Ideal se você precisa embutir essa funcionalidade em uma aplicação existente.

O Rodrigo Sendin explora todos os detalhes relativos ao acesso a banco de dados no Delphi for .NET, através do ADO.NET. Como engine de acesso, usa o provider do próprio Firebird para ADO.NET. Conheça o que são e para que servem os Managed Providers, componentes de conexão, FbDataReaders, FbCommands, FbDataAdapters e DataSets.

E ainda nesta edição: DataSnap, validações, serialização de objetos e muito mais!

Um abraço a todos, boa leitura e muito sucesso com o Delphi.

Guinther Pauli – guinther@devmedia.com.br
MCP, MCAD, MCSD.NET, Delphi Certified

ÍNDICE

08 - Explorando as classes de acesso a dados no ADO.NET
Rodrigo Sendin

14 - QA Audits - faça um ckeck-up no seu código-fonte
Marcos Alexandre Miguel e Marco Antônio Pereira Araújo

22 - Validação simplificada com os controles da JVCL
Fabricio Desbessel

27 - Usando Webcam com Delphi
Adriano Santos

32 - Configurando o Windows 2003 Server para aplicações DataSnap
Murilo Costa Monteiro Filho

36 - VoIP com Delphi - Desenvolva o seu próprio "Skype"
Jorge Luis Bublitz

40 - Automatizando backup no Firebird
Rafaela de Campos

44 - Serialização de objetos
Danilo Rocha Valente e Fernando Nomellini

48 - Criando uma ferramenta personalizada para backup no IB/FB
Rodrigo Otto Mostaert

Ask the Expert

Perguntas e Respostas

PChar?

O que é PChar? Vejo muito esse tipo de campo nos códigos Delphi, mas não sei o que posso fazer com ele.
Juliano Caldas

PChar nada mais é que um ponteiro para um *Array de Char*. O seu último caractere é um *Null*, *Chr(0)*, por isso também é chamado de *null-terminated string*. A primeira vista pode parecer simplesmente uma *string* que termina com *Chr(0)*, mas ao contrário da *string*, com *PChar* você deve tomar cuidado com alocação de memória, por ser um ponteiro.

Usamos esse tipo "esotérico" principalmente em duas situações: chamadas à *APIs* do Windows e chamadas a funções que estão em DLLs escritas em *C/C++*. Se a função pede um parâmetro do tipo *PChar* mas apenas lê seu conteúdo, você pode usar *strings* normalmente. Veja o exemplo a seguir, usando a função *MessageBox* do Windows:

```
procedure SayHello;
var
  sNormalString: string;
begin
  sNormalString:= 'Hello!!';
  MessageBox(0, PChar(sNormalString),
    'Constante String', 0);
end;
```

Se a função pede um *PChar* como parâmetro e altera o seu conteúdo, você deverá alocar memória. Após o retorno da função é muito fácil converter um *PChar* em *string*. Em *Delphi*, um *Array de Char* é compatível com *PChar*, com a vantagem de já alocar memória. Veja o exemplo a seguir, que retorna o caminho da pasta temp do Windows:

```
function MyGetTempPath: string;
var
  caminho: array[0..MAX_PATH] of Char;
  { MAX_PATH é uma constante do Windows }
begin
  GetTempPath(MAX_PATH, caminho);
  Result := caminho;
end;
```

Existem casos em que você não sabe o tamanho da memória a ser alocada. Nesses casos deve-se alocá-la dinamicamente. Veja o exemplo a seguir, que retorna o nome do usuário logado no Windows:

```
function MyGetUserName: string;
var
  nome: PChar;
  tam: DWORD;
begin
  nome := nil;
  { Pega o tamanho necessário para o Buffer }
  tam := 0;
  GetUserName(nome, tam);
  { Aloca memória (o #0 final é incluído em "tam") }
  nome := StrAlloc(tam);
  try
    { Chama a função novamente }
    GetUserName(nome, tam);
    Result := nome;
  finally { Limpa o buffer }
    StrDispose(nome);
  end;
end;
```

Resposta enviada por **Jorge Bublitz** (*bublitz@hotmail.com*).

Portal ClubeDelphi PLUS

+200 vídeo aulas

www.clubedelphi.net/portal

Caro Leitor,

O portal ClubeDelphi PLUS é a continuação, na Web, da revista ClubeDelphi. O portal recebe um conteúdo novo todo dia e hoje conta com: i) mais de 200 vídeo aulas; ii) diversos mini cursos gratuitos; iii) 1 livro eletrônico gratuito, de Guinther Pauli, sobre ADO.NET e BDP; iv) mais de 140 artigos exclusivos (que não foram publicados na revista)!;

Acesse o portal ClubeDelphi PLUS e receba muito mais conteúdo sobre Delphi! E o que é melhor: de graça! Todo leitor da revista ClubeDelphi, seja ele assinante ou comprador da revista em bancas, tem acesso ao portal (para quem compra em bancas, o acesso é válido por 30 dias).

Se você é assinante, utilize o seu login e senha pessoais para acessar o portal. Se você comprou em bancas, utilize o login e senha publicados na página do editorial desta edição.

Confira a seguir as últimas novidades do portal!

Boa leitura e sucesso!
Equipe DevMedia

Artigos

SQL Server - Criando banco, tabelas e acesso no Delphi - Parte I e II

Veja os artigos de Luciano Pimenta, que mostram como criar banco, tabelas e acesso ao SQL Server 2000 com o Delphi.

Curso de dbExpress e DataSnap - Parte XXIV a XXX

Veja os artigos finais do curso de Guinther Pauli, sobre dbExpress e DataSnap.

Vídeo-Aulas

Trabalhando com o IBExpert

Veja nessa vídeo aula de Luciano Pimenta, como trabalhar com a ferramenta de manutenção de bancos IB/FB.

Imprimindo os registros selecionados em um DBGrid com o QuickReport

Veja nessa vídeo aula de Luciano Pimenta, como imprimir registros selecionados em um DBGrid com o Quick Report.

Código de barras no Rave Reports

Veja nessa vídeo aula de Luciano Pimenta, como trabalhar com código de barras no Rave.

Minicurso SysCash - Parte IV

Veja nessa vídeo aula de Everson Volaco, mas um capítulo que mostra o sistema SysCash.

Imprimindo os registros selecionados no DBGrid com Rave

Veja nessa vídeo aula de Luciano Pimenta, como imprimir somente os itens selecionados em um DBGrid com o Rave.

Código de barras no Quick Report

Veja nessa vídeo aula de Luciano Pimenta, como trabalhar com código de barras no Quick Report.

Global Pages no Rave Reports

Veja nessa vídeo aula de Luciano Pimenta, como trabalhar com páginas globais no Rave.

Alterando relatórios no Rave em tempo de execução

Veja nessa vídeo aula de Luciano Pimenta, como dar a opção ao usuário alterar relatórios no Rave.

Trabalhando com Web Services no Delphi 2006

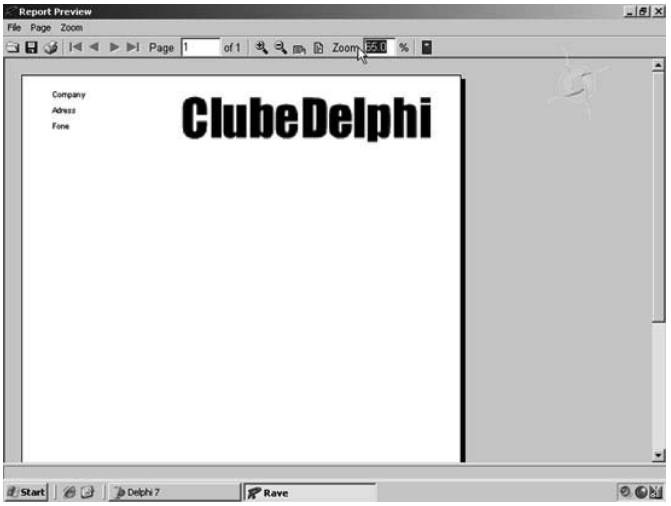
Veja nessa vídeo aula de Luciano Pimenta, como trabalhar com Web Services no Delphi 2006.

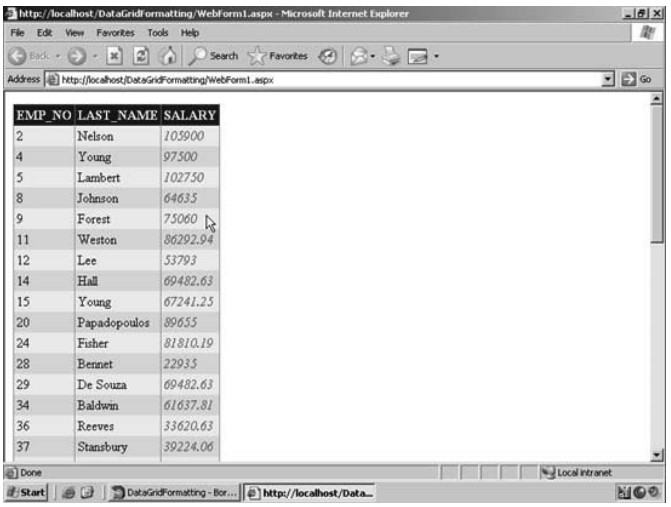
Mostrando imagens no DataGrid

Veja nessa vídeo aula de Luciano Pimenta, como mostrar imagens em um DataGrid do ASP.NET.

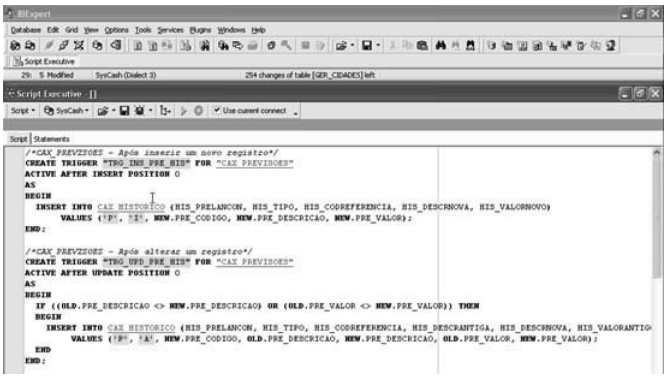
Preenchendo o DataGrid com informações de diretórios

Veja nessa vídeo aula de Luciano Pimenta, como listar arquivos e diretórios no DataGrid.



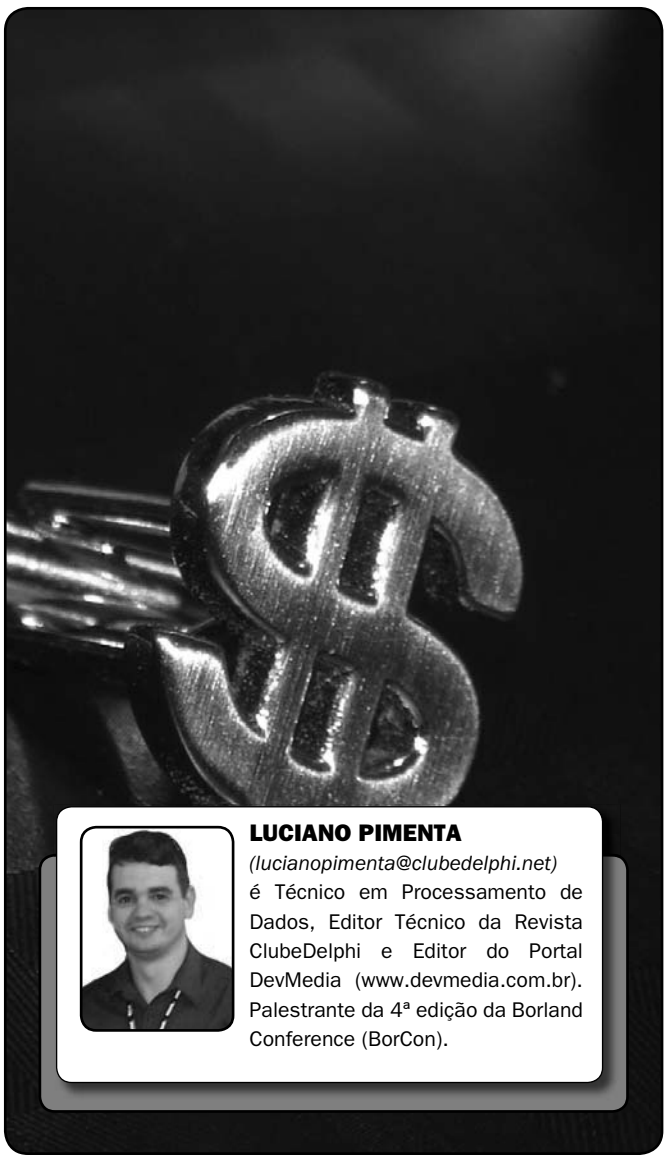


EMP_NO	LAST_NAME	SALARY
2	Nelson	105900
4	Young	97500
5	Lambert	102750
8	Johnson	64635
9	Forest	75060
11	Weston	86292.94
12	Lee	53793
14	Hall	69482.63
15	Young	67241.25
20	Papadopoulos	89655
24	Fisher	81810.19
28	Bennet	22935
29	De Souza	69482.63
34	Baldwin	61637.81
36	Reeves	33620.63
37	Stansbury	39224.06



+ de 80.000 membros cadastrados
+ de 15.000 exemplos com fontes
+ de 900 apostilas
+ de 4.000 dicas
Fórum Delphi
Artigos
TOTALMENTE GRÁTIS
www.delphi.eti.br
Um dos maiores sites de apoio a desenvolvedores Delphi do Brasil!!!

Boletos e cobrança com cartão de crédito



LUCIANO PIMENTA
(lucianopimenta@clubedelphi.net)
é Técnico em Processamento de Dados, Editor Técnico da Revista ClubeDelphi e Editor do Portal DevMedia (www.devmedia.com.br). Palestrante da 4ª edição da Borland Conference (BorCon).

Vimos na edição 77 da Clube Delphi, na última parte do mini-curso onde criamos uma aplicação Web completa usando ASP.NET, a solução CobreBem, que gera boletos bancários e realiza a cobrança de pagamentos via cartão de crédito.

Neste artigo veremos, como fazer essas funcionalidades em aplicações Win32, usando o Delphi 7. Imagine a valorização que o seu software teria ao embutir a geração de boletos bancários diretamente na aplicação, sem a necessidade de softwares bancários instalados no cliente. Vamos aos exemplos.

Baixando e instalando o CobreBemX

Existe uma versão para desenvolvimento e avaliação da ferramenta no site do desenvolvedor (veja seção *Links*). Com essa versão, existem duas restrições: a data de vencimento é fixa (com o ano de 2007), e o valor cobrado é mostrado como R$ 1,90.

Baixe a versão de teste no endereço: *www.cobrebem.com/downloads/cbx/InstalarCobreBemX.zip* e descompacte o arquivo em uma pasta qualquer. Clique no arquivo *Instalar.exe* para registrar a DLL. O controle é um componente COM/DCOM e não será instalado no IDE do Delphi 7.

Existem mais algumas configurações e arquivos a serem baixados para gerar corretamente os boletos. Acesse o endereço *www.cobrebem.com/downloads/cbx/ImagensBoleto.zip* para baixar as imagens do boleto, que devem ser descompactadas na pasta da aplicação que vamos criar (de o nome de "imagens-boleto" para a pasta).

Não é recomendado alterar o nome dessas imagens, pois as mesmas serão usadas para o logotipo do banco, bem como o código de barras. Devemos no final, gerar um arquivo, que será a licença de teste e o tipo de cobrança do boleto.

Acesse o endereço *www.cobrebem.com/cgi-bin/GeraArquivoLicencaTeste*, escolha o banco (existem vários) e em *Carteira*,

escolha uma cobrança simples. Clique em *Gerar Arquivo* e salve o arquivo na pasta da aplicação (**Figura 1**).

> **Nota:** Para esse artigo, foi gentilmente fornecida uma licença do componente, então os valores mostrados no boleto serão os valores reais da aplicação.

Usando o CobreBemX no Delphi Win32

Como comentando anteriormente, o CobreBem não instala nenhum componente no IDE do Delphi, devemos trabalhar com objetos OLE. Crie uma aplicação no Delphi 7 e adicione os componentes da **Figura 2**.

Vamos simular um cadastro de parcelas, onde vamos gerar um boleto bancário para cada parcela e realizar o pagamento via cartão de crédito, com a quantidade de parcelas. Não criaremos um banco de dados, pois despenderia um trabalho em termos de tabelas e relacionamentos, portanto usarei apenas um *ClientDataSet* em memória para gerar as parcelas. A idéia serve para ser usada em uma aplicação real, claro. Adicione os campos da **Tabela 1** no *ClientDataSet*.

Clique com o botão direito do mouse no *ClientDataSet* e escolha *Create DataSet*. Para o botão *Gerar Parcelas*, adicione o código da **Listagem 1**.

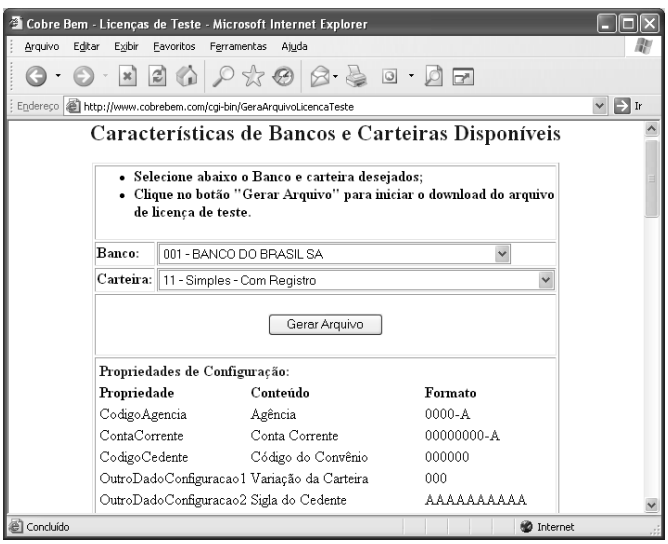


Figura 1. Gerando o arquivo de licença de teste e o tipo de carteira

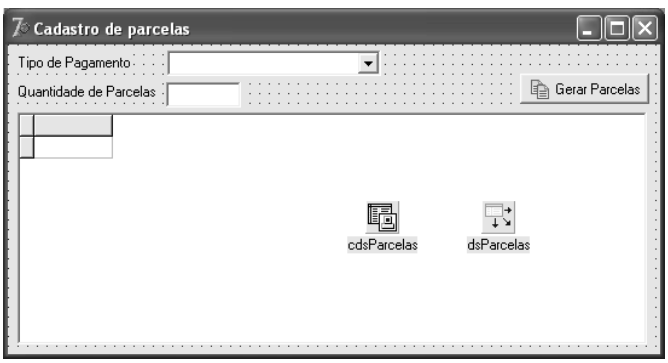


Figura 2. Tela de cadastro de parcelas da aplicação

Listagem 1. Código para gerar as parcelas

```
var
  i: integer;
  aDtVencimento: TDateTime;
begin
  aDtVencimento := Date + 30;
  for i := 1 to StrToInt(Edit1.Text) do
  begin
     with cdsParcelas do
     begin
       Append;
       FieldByName('NUMERO').AsInteger := i;
       FieldByName('NOME').AsString := 'Fulano de Tal';
       FieldByName('CPF').AsString := '11111111111';
       FieldByName('ENDERECO').AsString := 'Rua do Sacado';
       FieldByName('BAIRRO').AsString := 'Bairro do Sacado';
       FieldByName('CIDADE').AsString := 'Cidade do Sacado';
       FieldByName('CEP').AsString := '97500000';
       FieldByName('UF').AsString := 'RS';
       FieldByName('DATA_DOCUMENTO').AsDateTime := Date;
       if i <> 1 then
         aDtVencimento := IncMonth(aDtVencimento, 1);
       FieldByName('DATA_VENCIMENTO').AsDateTime := aDtVencimento;
       FieldByName('VALOR').AsCurrency := 171.50;
       Post;
     end;
  end;
```

Se quiser testar, execute a aplicação e gere a quantidade de parcelas desejada. Vamos agora adicionar código para gerar o boleto bancário. Declare na seção *uses* a unit *ComObj* e na seção *public* declare as seguinte variáveis:

```
CobreBemX: Variant;
Boleto: Variant;
```

Crie um método chamado *GeraBoleto* e implemente-o com o código da **Listagem 2**.

Listagem 2. Gerando o boleto bancário

```
procedure TForm1.GeraBoleto;
begin
  CobreBemX := CreateOleObject('CobreBemX.ContaCorrente');
  CobreBemX.ArquivoLicenca := ExtractFileDir(
    Application.ExeName)+ '\001-11.conf';
  CobreBemX.CodigoAgencia := '1234-5';
  CobreBemX.NumeroContaCorrente := '00000123-X';
  CobreBemX.CodigoCedente := '123456';
  CobreBemX.InicioNossoNumero := '00001';
  CobreBemX.FimNossoNumero := '99999';
  CobreBemX.ProximoNossoNumero := '00015';
  CobreBemX.PadroesBoleto.PadroesBoletoImpresso.
    ArquivoLogotipo := ExtractFileDir(
    Application.ExeName)+'\logoboleto.gif';
  CobreBemX.PadroesBoleto.PadroesBoletoImpresso.
    CaminhoImagensCodigoBarras := ExtractFileDir(
    Application.ExeName)+'\imagensboleto\';
  cdsParcelas.First;
  while not cdsParcelas.Eof do
  begin
    Boleto := CobreBemX.DocumentosCobranca.Add;
    Boleto.NumeroDocumento :=cdsParcelasNUMERO.AsString;
    Boleto.NomeSacado := cdsParcelasNOME.AsString;
    Boleto.CPFSacado := cdsParcelasCPF.AsString;
    Boleto.EnderecoSacado := cdsParcelasENDERECO.AsString;
    Boleto.BairroSacado := cdsParcelasBAIRRO.AsString;
    Boleto.CidadeSacado := cdsParcelasCIDADE.AsString;
    Boleto.EstadoSacado := cdsParcelasUF.AsString;
    Boleto.CepSacado := cdsParcelasCEP.AsString;
    Boleto.DataDocumento := cdsParcelasDATA_DOCUMENTO.AsDateTime;
    Boleto.DataVencimento := cdsParcelasDATA_VENCIMENTO.AsDateTime;
    Boleto.ValorDocumento:= cdsParcelasVALOR.AsFloat;
    Boleto.PadroesBoleto.Demonstrativo :=
      'Referente parcelamento na Loja DevMedia<br>'+
      '<b>Parcela número: '+ FormatFloat('0000',
      cdsParcelasNUMERO.AsInteger) + '</b>';
    Boleto.PadroesBoleto.InstrucoesCaixa :=
      '<br><br>Não dispensar juros e multa após o '+
      'vencimento';
    cdsParcelas.Next;
  end;
  CobreBemX.Imprimeboletos;
  CobreBemX := Unassigned;
end;
```

Campo	Tipo	Tamanho	Descrição
ID	AutoInc	–	Campo chave da parcela.
NUMERO	Integer	–	Número da parcela.
NOME	String	35	Nome do Cliente (em uma aplicação real, esse campo seria oriundo de consulta ao banco. Teríamos no caso, apenas o código do cliente).
CPF	String	11	CPF do Cliente
ENDERECO	String	30	Endereço do Cliente
BAIRRO	String	25	Bairro do Cliente
CIDADE	String	20	Cidade do Cliente
CEP	String	8	CEP do Cliente
UF	String	2	Estado
DATA_DOCUMENTO	Date	–	Data do documento
DATA_VENCIMENTO	Date	–	Data de vencimento da parcela
VALOR	Currency	–	Valor da parcela

Tabela 1. Campos no ClientDataSet da aplicação

O método preenche os dados comuns do boleto, a seguir percorremos o *ClientDataSet* para adicionar os dados de cada boleto e por fim, chamamos *ImprimeBoletos*. No *OnClick* do botão, após o laço de inserção dos dados, adicione o seguinte código:

```
...
if ComboBox1.ItemIndex = 0 then
  GeraBoleto;
```

Execute a aplicação, digite a quantidade de parcelas e clique no botão para visualizar os boletos (**Figura 3**).

Se quiser, compare com o código da edição 77 que gerou boleto em ASP.NET e veja que o código é praticamente o mesmo. Temos também a possibilidade de envio do boleto por e-mail, com o código praticamente igual ao mostrado na edição da aplicação Web completa.

Pagamento com cartão de crédito

Para este exemplo, vamos criar um formulário para digitação das informações de consulta (número do cartão, validade etc.). Crie um novo formulário dando o nome de "frmcartao". Adicione os componentes mostrados na **Figura 4** (note o nome de cada *Edit*).

O arquivo CONF gerado anteriormente, pode ser usado para a cobrança de cartão de crédito, mas a consulta de cartão quase sempre retorna como não aprovado (claro, estamos trabalhan-

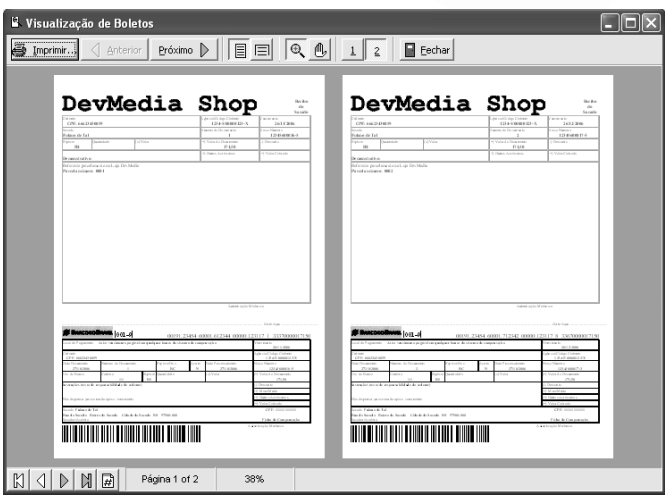


Figura 3. Boletos gerados pela aplicação Win32

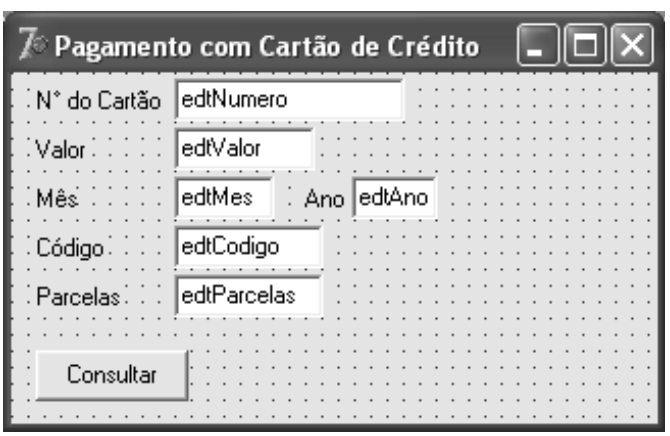


Figura 4. Criando o formulário para consulta do cartão de crédito

ClubeDelphi PLUS

Acesse agora mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como criar boletos bancários com o CobreBem no Delphi 7.

www.devmedia.com.br/articles/viewcomp.asp?comp=2714

do com testes e o mesmo é feito em ordem aleatória).

Você pode baixar o *AprovaFácil* (outra solução para venda com cartão de crédito) e usar o *cartaoteste.conf* presente no arquivo zipado, bastando colocá-lo no diretório do projeto (não esqueça de alterar o nome do arquivo na respectiva propriedade).

Declare na seção *uses* a unit *ComObj*. Para o botão *Consultar*, adicione o código da **Listagem 3**.

Listagem 3. Consulta do cartão de crédito

```
var
  CobreBemX: Variant;
  Boleto: Variant;
  msg: string;
begin
  CobreBemX := CreateOleObject('CobreBemX.ContaCorrente');
  { arquivo de licença }
  CobreBemX.ArquivoLicencaCartoesCredito :=
    ExtractFileDir(Application.ExeName)+
    '\cartaoteste.conf';
  { 0 = Teste, 1 = Simulação e 2 = Produção }
  CobreBemX.PadroesAprovacaoCartoes.ModoOperacao := 0;
  { Cria documento de cobrança e monta dados do cartão }
  CobreBemX.DocumentosCobranca.Clear;
  Boleto := CobreBemX.DocumentosCobranca.Add;
  { repassa os valores dos controles }
  Boleto.CartaoCredito.Numero := edtNumero.Text;
  Boleto.ValorDocumento := StrToFloat(edtValor.Text);
  Boleto.CartaoCredito.MesValidade := StrToInt(edtMes.Text);
  Boleto.CartaoCredito.AnoValidade := StrToInt(edtAno.Text);
  Boleto.CartaoCredito.QuantidadeParcelas := StrToInt(edtParcelas.Text);
  Boleto.CartaoCredito.CodigoSeguranca := edtCodigo.Text;
  { solicita a aprovação }
  if CobreBemX.SolicitaAprovacaoCartao[0] then
  begin
     { aprovado }
     msg := Boleto.CartaoCredito.ResultadoSolicitacaoAprovacao;
     { confirma a aprovação }
     CobreBemX.ConfirmaAprovacaoCartao[0];
     { pega o resultado da aprocação }
     msg := msg + #13#10 + Boleto.CartaoCredito.
       ResultadoSolicitacaoAprovacao + #13#10 +
       Boleto.CartaoCredito.ComprovanteAdministradora;
  end
  else
     { não aprovado }
    msg := Boleto.CartaoCredito.ResultadoSolicitacaoAprovacao;
  { exibe o texto da aprovação }
  MessageDlg(msg, mtInformation, [mbOK], 0);
  CobreBemX.DocumentosCobranca.Clear;
  CobreBemX := Unassigned;
end;
```

Note que temos um "modo de operação" que pode ser Teste, Simulação ou Produção, sendo que podemos testar à vontade usando o *moacTeste*. Os dados referente ao valor total, pode ser preenchido com um campo *Aggregate* do *ClientDataSet* e repassado entre os formulários.

No formulário principal altere o código do botão de geração de parcelas, com o seguinte código:

```
...
  GeraBoleto
else
  frmcartao.ShowModal;
```

Execute a aplicação, escolha o tipo de pagamento e gere as parcelas. Veja na **Figura 5** as mensagens emitidas.

Os dados do cartão mostrados na figura anterior são de testes.

Conclusão

Vimos que podemos trabalhar com boletos bancários em Win32 de forma ágil e rápida, sem a necessidade de criarmos nossos próprios boletos, que dependeria de um esforço muito grande de desenvolvimento.

A cobrança via cartão de crédito tornou-se cada vez mais comum para empresas, sendo assim, seu software deve se adequar às novidades e necessidades dos clientes. Um grande abraço a todos e até a próxima! ■

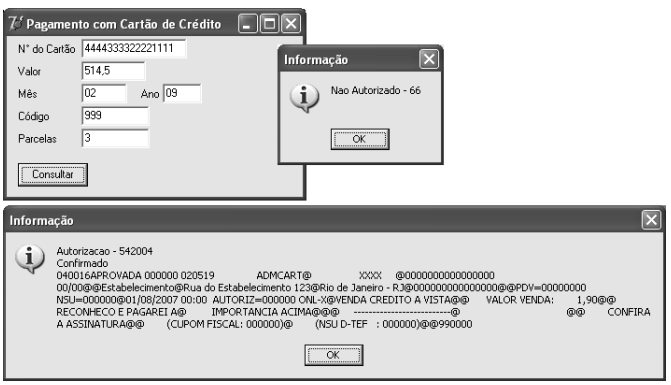


Figura 5. Mensagens de aprovação ou não da consulta

Links

CobreBem

www.cobrebem.com.br

ClubeDelphi PLUS

Acesse agora mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Everson Volaco que mostra como trabalhar com campos Aggregates no ClientDataSet.

www.devmedia.com.br/articles/viewcomp.asp?comp=1376

ClubeDelphi PLUS

Acesse agora mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como realizar pagamentos com cartão de crédito com o CobreBem no Delphi 7.

www.devmedia.com.br/articles/viewcomp.asp?comp=2740

ClientDataSet

Técnicas Avançadas

CARLOS EDUARDO BIER
(eduardo.ceb@hotmail.com)
é programador e desenvolvedor, atua há mais de 10 anos na área de informática, sendo os últimos 5 dedicados ao Delphi. Participou e participa constantemente de vários treinamentos e eventos oficiais Borland em Curitiba – PR.

Estamos acostumados a utilizar uma gama de recursos cada vez mais interessantes e úteis que vem sendo disponibilizados a cada versão da VCL. Porém, como nem tudo é perfeito, ainda assim nos deparamos com situações em que não encontramos soluções para problemas relativamente simples. Vou relatar um caso que vivenciei.

Certa vez, quando participava do desenvolvimento de um software, optei por utilizar em cada módulo do sistema um formulário inicial onde o usuário pudesse inserir ou, através de vários parâmetros para filtragem, localizar um registro para alteração. Esse formulário por sua vez chamava um segundo que trabalhava totalmente em memória, trazendo os campos do registro preenchidos ou vazios em caso de inserção.

Se houvesse confirmação da operação, era feita a validação e envio ao banco de dados, caso contrário os dados eram descartados. No caso de formulários com informações mestre/detalhe, utilizava um *DBGrid* ligado a um *ClientDataSet* para apresentação dos detalhes. Esse era alimentado por uma *procedure* de busca que montava em tempo de execução uma consulta SQL para retornar os detalhes de acordo com o código do registro mestre passado como parâmetro e, através de um *while* inseria os registros no *ClientDataSet*.

Certo, boa idéia para trabalhar os dados em memória antes do envio ao banco. No entanto, surgiu o seguinte problema: quando estamos alterando um registro que tinha detalhes e esses estando carregados no *ClientDataSet*, no momento de salvar as informações, a *procedure* responsável pela gravação passava pelos detalhes verificando seu *UpdateStatus* para definir qual instrução SQL executar.

O retorno era *usInserted* mesmo que os registros não tivessem sofrido nenhuma modificação pelo usuário. Ao tentar inserir no banco de dados uma exceção de violação de chave era levantada. Agora chegamos a um problema que poderia ser facilmente solucionado. Bastava que no momento de carregar

os detalhes, após inserir o registro no *ClientDataSet*, tivéssemos um recurso do componente para voltar seu estado para *usUnModified*, porém não temos. O que fazer?

Funcionamento interno do ClientDataSet

Como todo problema deve ser resolvido, mãos a obra! Minha saída foi estudar o funcionamento do *ClientDataSet* para encontrar uma maneira de manipular o estado dos registros. Analisando a classe *TClientDataSet* descobri que ela somente publica as propriedades, métodos e eventos de sua ancestral, *TCustomClientDataSet* onde todo código é implementado.

Percebam no código a seguir, que somente o escopo *published* é utilizado:

```
TClientDataSet = class(TCustomClientDataSet)
  published
    property Active;
    property Aggregates;
...
```

O próximo passo foi verificar o funcionamento da função *UpdateStatus*. Pude perceber que todo registro tem um ponteiro que é controlado internamente pelo componente, apontando para um tipo declarado que contém informações internas sobre o mesmo.

Entre essas informações, o campo *Attribute* que é do tipo *DSAttr (byte)* declarado em *DBIntF* e que assume valores de algumas constantes que indicam o estado atual do registro no *DataSet*. Através da análise desse campo (**Listagem 1**), a função *UpdateStatus* retorna o valor correspondente.

A solução

Estava perto, já sabia como a função *UpdateStatus* trabalha para retornar o estado do registro, agora tinha que aprender alterar seu *Attribute* internamente para chegar ao resultado desejado. Confesso que a primeira solução que me veio à cabeça, foi de adicionar uma *procedure* no próprio *TCustomClientDataSet* e publicar na classe *TClientDataSet*.

Cheguei a implementar, porém não achei muito elegante, sem falar no inconveniente da necessidade de recompilar o componente toda vez que reinstalava o Delphi. Então pensei em fazer algo mais profissional.

Voltei ao mundo real, minha aplicação. Pensei: "Preciso alterar uma informação interna e protegida do componente". Cheguei a conclusão que teria que utilizar o recurso de *Classes amigas*, não haveria outra solução.

Essa técnica, apesar de não recomendada, permite o acesso a informações declaradas no escopo *private* e *protected* de classes ascendentes, em nosso caso permitirá o acesso a campos da *TCustomClientDataSet*. A partir da versão 2005 do Delphi, foram incluídos dois novos escopos de visibilidade: *strict private* e *strict protected*, que permitem bloquear a utilização desse recurso caso você tenha necessidade. Para nossa alegria a VCL não foi escrita utilizando os novos escopos.

Precisei declarar uma classe descendente de *TCustomClientDataSet*, porém com um detalhe: é necessário colocar no nome da unit o tipo que vamos descender, em nosso caso *DBClient*. *TCustomClientDataSet*. Utilizando essa nova classe, bastou escrever uma *procedure* que altere o valor interno de *Attribute* e resolvemos nosso problema (**Listagem 2**).

Com a *procedure* implementada, onde você julgar conveniente em sua aplicação, basta fazer a chamada passando como parâmetro o *ClientDataSet* e qual valor de *UpdateStatus* queremos setar.

CloneCursor

Certo dia no ambiente de trabalho entrei em uma "discussão" com os demais analistas sobre o uso do recurso *CloneCursor*, alguns defendendo e outros criticando. A partir daí, resolvi fazer um estudo no método e utilizá-lo para analisar se existe algum ponto negativo ao fazer uso do recurso.

Algo que foi exposto, seria que supostamente o método faria uma nova cópia do conteúdo do *ClientDataSet* em memória, sendo essa uma das desvantagens do uso, porém analisando o

Listagem 1. Código analisado da função UpdateStatus

```
unit DBClient;

interface
  {$IFDEF MSWINDOWS}
  uses Windows, SysUtils, VarUtils, Variants, Classes,
    DB, DSIntf, DBCommon, DBCommonTypes, Midas,
    SqlTimSt, ActiveX;
  {$ENDIF}
  {$IFDEF LINUX}
  uses Libc, SysUtils, VarUtils, Variants, Classes,
    DB, DSIntf, DBCommon, Midas, SqlTimSt;
  {$ENDIF}

type
  ...
  ...
  PRecInfo = ^TRecInfo;
  TRecInfo = packed record
    RecordNumber: Longint;
    BookmarkFlag: TBookmarkFlag;
    Attribute: DSAttr;
  end;
  ...
  ...
TCustomClientDataSet = class(TWideDataSet)
  private
    ...
    ...
  public
    function UpdateStatus: TUpdateStatus; override;
  end;

implementation

function TCustomClientDataSet.UpdateStatus:
  TupdateStatus;
var
  BufPtr: Pchar;
  Arrt: Byte;
begin
  CheckActive;
  if State = dsInternalCalc then
    Result := usUnModified
  else
  begin
    if State = dsCalcFields then
      BufPtr := CalcBuffer else
      BufPtr := ActiveBuffer;
    Attr := PRecInfo(BufPtr + FRecInfoOfs).Attribute;
    if (Attr and dsRecModified) <> 0 then
      Result := usModified
    else if (Attr and dsRecDeleted) <> 0 then
      Result := usDeleted
    else if (Attr and dsRecNew) <> 0 then
      Result := usInserted
    else
      Result := usUnModified;
  end;
end;
end.
```

código percebi que isso não ocorre. Na verdade o que ocorre é que, quando invocamos *CloneCursor* as propriedades internas do componente passam a apontar para o *Source* que foi passado como parâmetro, ou seja, o *ClientDataSet* utiliza exatamente os mesmos dados do componente de origem somente com o cursor independente.

Então temos assim um "espelho" dos dados em um novo componente, resumidamente temos um *ClientDataSet* com nossos dados com dois cursores. Esse recurso é bastante interessante quando necessitamos buscar informações de outros registros para trabalhar com o registro corrente no *ClientDataSet*.

Para mostrar de forma clara a utilidade e o funcionamento do recurso, montarei um exemplo.

Exemplo 1. Acumulando valores em coluna

Vamos criar uma aplicação que contenha um *ClientDataSet* trabalhando em memória com três colunas: "Id" (*integer*), "Valor" (*currency*) e "Acumulado" (*currency*). Ao inserir um novo registro, a coluna *Acumulado* armazenará automaticamente os valores digitados na coluna *Valor*, acumulando com o valor do registro anterior.

Para isso, ao inserir um novo registro, no evento *BeforePost* do *ClientDataSet*, vamos utilizar um "espelho" do *ClientDataSet* para buscar o valor do registro anterior, já que, o cursor está posicionado no novo registro que estamos inserindo.

Vamos ao trabalho, crie uma nova aplicação e adicione um *ClientDataSet* contendo os campos citados anteriormente. Adicione um *DataSource* e um *DBGrid* para que possamos visualizar o efeito do código que será escrito. Faça a ligação entre os componentes.

No evento *BeforePost* do *ClientDataSet* insira o código da **Listagem 3**.

Veja na **Figura 1** o exemplo em execução

No momento da inserção do registro, fazemos uma busca no mesmo *DataSet*, procurando pelo último valor inserido e controlado pelo campo ID que deve ser preenchido, simulando um campo auto-incremento, com valores sequenciais. A coluna *Acumulado* automaticamente acumulará os valores de forma bastante simples, sem a necessidade de declaração de variável pública na unit, por exemplo.

Podemos então perceber que temos uma cópia fiel do *ClientDataSet Source* somente com um cursor independente.

Figura 1. Acumulando o valor da coluna, usando o CloneCursor

Listagem 2. Problema resolvido

```
unit UFormExemplo;

interface
  uses
    Windows, Messages, SysUtils, Variants, Classes,
    Graphics, Controls, Forms, Dialogs, DB, DBClient,
    StdCtrls, DSIntF, DBXpress, SqlExpr;

type
  TCDSAmiga = class(DBClient.TCustomClientDataSet);
  private
    procedure SetarEstadoDeRegistro(
      Cds: TClientDataSet;
      UpdateStatus: TUpdateStatus);
...

implementation

{$R *.dfm}

procedure TFormExemplo.SetarEstadoDeRegistro(
  Cds: TClientDataSet; UpdateStatus: TUpdateStatus);
var
  RecInf: Integer;
begin
  { Verifica se o ClientDataSet passado como
    parâmetro está ativo }

  if (not Cds.Active) then
    begin
      raise Exception.Create(
        'Erro ao alterar o estado do registro');
      Exit;
    end;

  { Recupera o OffSet do registro }
  RecInf := TCDSAmiga(Cds).GetRecordSize +
    TCDSAmiga(Cds).CalcFieldsSize;

  { Altera o Attribute de acordo com parâmetro passado }
  case UpdateStatus of

    usUnmodified: PRecInfo(TCDSAmiga(Cds).ActiveBuffer+
      RecInf).Attribute := dsRecUnmodified;

    usModified: PRecInfo(TCDSAmiga(Cds).ActiveBuffer+
      RecInf).Attribute := dsRecModified;

    usInserted: PRecInfo(TCDSAmiga(Cds).ActiveBuffer +
      RecInf).Attribute := dsRecNew;

    usDeleted: PRecInfo(TCDSAmiga(Cds).ActiveBuffer +
      RecInf).Attribute := dsRecDeleted;
  end;

  { Efetiva a alteração no Log de Alterações do
    componente }
  Cds.MergeChangeLog;
end;
end.
```

Listagem 3. Clonando o ClientDataSet

```
procedure TForm1.ClientDataSet1BeforePost(
  DataSet: TDataSet);
var
  cdsTmp: TClientDataSet;
begin

  try
    { Cria o objeto ClientDataSet que será o Clone do
      ClientDataSet da tela }
    cdsTmp := TClientDataSet.Create(Self);
    { Invoca o CloneCursor para gerar um "Clone" do
      ClientDataSet passado como source }
    cdsTmp.CloneCursor(ClientDataSet1, False);
    { Nesse ponto, localiza o último ID para buscar o
      valor do último campo acumulado }
    cdsTmp.Locate('ID', ClientDataSet1ID.AsInteger-1,
      [loCaseInsensitive]);
    { O Campo "Acumulado" recebe o valor do último
      registro + o valor digitado na coluna Valor }
    ClientDataSet1Acumulado.AsCurrency :=
      cdsTmp.FieldByName('Acumulado').AsCurrency +
      ClientDataSet1.FieldByName('Valor').AsCurrency;
  finally
    { O objeto utilizado para "espelhar" o
      ClientDataSet é destruído }
    FreeAndNil(cdsTmp);

  end;
end;
```

Exemplo 2. Apagando um range de registros

Ainda na mesma aplicação, faremos uma rotina para executar a tarefa de excluir um *range* de registros, apagando todos de uma só vez. Adicione um botão no formulário e no evento *OnClick* digite o código da **Listagem 4**.

Listagem 4. Apagando os registros

```
procedure TForm1.Button1Click (Sender: TObject);
var
  cdsTmp: TClientDataSet;
begin
  try
    { Cria o ClientDataSet que será o clone }
    cdsTmp := TClientDataSet.Create(Self);
    { Invoca o CloneCursor para gerar um "Clone" do
      ClientDataSet passado como source. Utiliza o
      parâmetro Reset como True para limpar filtros
      aplicados no source }
    cdsTmp.CloneCursor(ClientDataSet1, True);
    { Indica qual campo será utilizado como índice
      para o range }
    cdsTmp.IndexFieldNames := 'ID';
    { Passa os parâmetros indicando os Ids entre
      2 e 5 }
    cdsTmp.SetRange([2],[5]);
    { Apaga os registros do CDS Clone }
    while cdsTmp.RecordCount > 0 do
      cdsTmp.Delete;
  finally
    cdsTmp.Free;
  end;
end;
```

Na **Figura 2**, temos cinco registros no *ClientDataSet*.

Ao clicar no botão, todos que estão incluídos no *range* especificado são apagados e refletidos automaticamente no *ClientDataSet* ligado ao *DBGrid* (**Figura 3**).

É importante, claro, entender os parâmetros do *CloneCursor*, listados a seguir:

- *Source*: componente do tipo *TClientDataSet* que será a origem dos dados que serão clonados no *ClientDataSet* que invoca o método;
- *Reset* e *KeepSettings*: são utilizados para controlar se algumas propriedades e eventos serão também clonados do *ClientDataSet Source*, ou se no momento da cópia, essas serão mantidas como *default*, zeradas. As propriedades são: *Filter, Filtered, FilterOptions* e o evento *OnFilterRecord*.

Figura 2. Cinco registros adicionados em memória

Figura 3. Registros apagados no "clone", refletem no ClientDataSet da aplicação

ClubeDelphi PLUS

Acesse agora mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como usar o ClientDataSet como um vetor de dados, usando somente informações de memória.

www.devmedia.com.br/articles/viewcomp.asp?comp=555

Conclusão

Apesar da complexibilidade envolvida no estudo do *ClientDataSet*, vimos que podemos ter acesso às propriedades protegidas do mesmo. Podemos perceber também que o recurso de *CloneCursor* pode ser utilizado para automação de várias tarefas e, ao contrário da dúvida inicial, concluimos que não existirá um aumento de uso da memória, ou seja, uma nova alocação para se trabalhar com os dados clonados.

Só vale ressaltar que devemos atentar para os parâmetros passados e lembrar que toda e qualquer alteração feita no *ClientDataSet* clonado será refletida instantaneamente no *ClientDataSet* original.

Espero que tenham gostado das soluções por mim desenvolvida. Abraços e até a próxima. ■

Comparando estruturas de banco de dados IB/FB

EVERSON BORGES VOLACO
(everson@rhealeza.com.br)
é desenvolvedor e instrutor certificado Borland, com experiência em aplicações cliente/servidor, usando Delphi, Interbase e Oracle. Possui três certificações oficiais Borland: Borland Delphi 7.0, Borland CaliberRM 6.0 e Borland StarTeam 6.0.

Neste artigo conheceremos a ferramenta *DB Comparer for InterBase & Firebird*. Criada pela empresa *EMS Database Management Solutions* (*www.sqlmanager.net*) e atualmente na versão 2.2 (até o fechamento deste artigo).

O *DB Comparer* realiza comparações entre bancos de dados InterBase/Firebird, possibilitando assim visualizar diferenças em suas estruturas e gerar *scripts* para eliminar essas diferenças ou apenas as selecionadas.

O *DB Comparer* traz uma interface totalmente visual e de fácil uso. Além dessa interface, está disponível para o usuário uma aplicação console que possibilita a utilização de suas funcionalidades através de linha de comando.

> **Nota:** Você pode baixar a versão *trial* disponível para avaliação por 30 dias, através do site *www.sqlmanager.net/products/ibfb/dbcomparer*. O arquivo possui apenas 4,5 MB aproximadamente. É necessário ser cadastrado (gratuito) no site para baixar as versões *trial*.

Principais Características

O *DB Comparer 2006 for InterBase & Firebird* possui diversas funcionalidades úteis para a realização de comparações entre banco de dados InterBase/Firebird. Entre as principais encontram-se:

- Comparação entre bancos de dados armazenados em diferentes servidores;
- Suporte ao InterBase 7.5 e Firebird 2.0;
- Editor para scripts SQL com *sintax highlight*;
- Criação de *templates* para facilitar/acelerar edição de scripts SQL;
- Criação/modificação de esquemas visuais (*Visual*

Scheme) para customização do IDE;
- Designer visual para criação de relatórios e formulários;
- Permite a navegação sincronizada entre os objetos dos dois bancos de dados comparados;
- Permite a sincronização entre os bancos de dados comparados;
- Opção disponível para conexão através de canal SSH;
- Opção para visualização apenas dos objetos diferentes;
- Arquivo de ajuda, no formato CHM, integrado ao IDE.

Instalação

Após realizar o download, descompacte o arquivo e execute *IbComparer.exe* para iniciar a instalação da ferramenta. A instalação do *DB Comparer* é extremamente simples, de forma que não entrarei em detalhes aqui.

No diretório de instalação, o arquivo *IbComparer.exe* corresponde a ferramenta gráfica do *DB Comparer* enquanto que o *IbComparerC.exe* diz respeito à aplicação console, por linha de comando. Além dos arquivos executáveis você pode ter acesso ao arquivo de ajuda da ferramenta (*IbComparer.chm*).

Para iniciar o *DB Comparer* acesse a opção *DB Comparer 2006 for InterBase & Firebird* disponível a partir do menu Iniciar do Windows. Através desse menu de atalho temos acesso ainda à aplicação console e ao arquivo de ajuda.

Conhecendo o IDE

O *DB Comparer* trabalha com o conceito de projetos, isso é, para realizar a comparação entre dois bancos de dados IB/FB, você precisa criar um projeto dentro da ferramenta. Inicie o *DB Comparer* (interface gráfica) e clique na opção *File>New Project* disponível no menu principal. Como comentado anteriormente, o *DB Comparer* permite a comparação entre bancos de dados armazenados em diferentes servidores.

Para isso, basta configurar a propriedade *Server* para a seção *Master Database* e *Target Database*. Para demonstrar as funcionalidades da ferramenta utilizei o banco de dados *Employee.fdb* que acompanha o Firebird 1.5.2. Para realizar a comparação entre dois bancos, realizei uma cópia do banco *Employee* e alterei a sua estrutura de objetos.

Neste exemplo ambos os bancos de dados estão disponíveis no mesmo servidor Firebird. Veja na **Figura 1** a configuração realizada na janela *Project Options* para o novo projeto.

> **Obs:** Por padrão o campo *Library* vem definido como *gds32.dll* que é a biblioteca utilizada pelo InterBase. Para o Firebird altere o campo para *fbClient.dll*.

Você pode testar a conexão com o banco de dados através da opção *Test Connect*. Marcando a opção *Both databases on the same server* fazemos com que as configurações do servidor *Master* seja replicado para o *Target* visto que os dois bancos de dados residem no mesmo servidor.

A versão 2006 do *DB Comparer* possui uma nova opção, *Synchronize databases after comparing in console mode* que permite a sincronização automática entre os dois bancos de dados após a comparação dos mesmos. Caso essa opção seja marcada, podemos definir a direção da sincronização através do campo *Direction of synchronization*.

Selecionado a aba *Compare Options* podemos definir que objetos serão comparados assim como as propriedades/campos de cada um. Através da seção *Generate scripts for* podemos definir ainda quais tipos de *scripts* queremos gerar para cada tipo de objeto de banco de dados.

Em resumo a aba *Compare Options* permite parametrizar as comparações a serem realizadas assim como definir filtros e quais *scripts* gerar (**Figura 2**).

Por último temos a aba *Comment* que permite digitar com um texto, descrevendo o propósito da comparação entre os bancos nesse projeto. Clique no botão OK para iniciar o processo de comparação. Esse processo consiste na conexão aos bancos de dados informados e a comparação de cada objeto

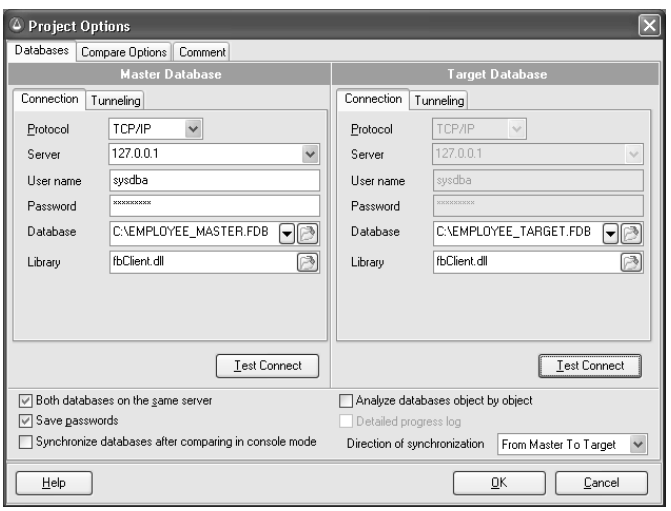


Figura 1. Configurando o acesso aos bancos de dados a serem comparados

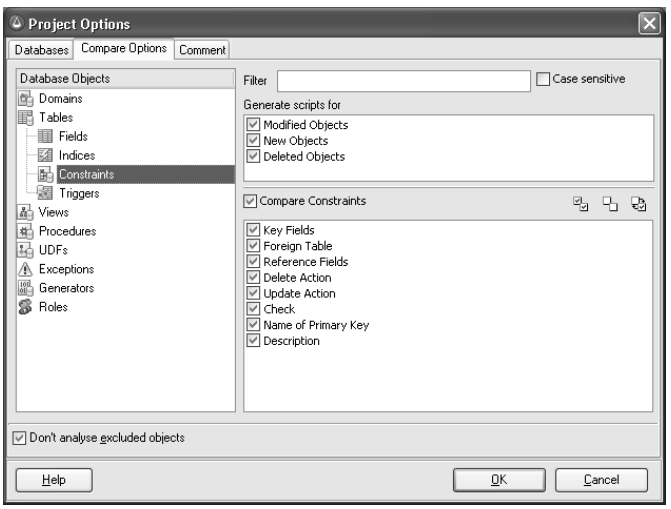


Figura 2. Definindo os objetos e propriedades a serem comparadas

disponível nos mesmos.

Ao terminar o processo clique no botão *Close* na janela *Progress*. Antes de analisarmos o resultado da comparação salve o projeto através da opção *File>Save Project*. Será gerado um arquivo de extensão ICP que nada mais é do que um arquivo INI contendo as configurações definidas no projeto.

Dica: Para verificar o conteúdo do arquivo ICP e a forma como a ferramenta armazena as informações do projeto, abra o arquivo no Bloco de Notas do Windows.

Após a comparação, o *DB Comparer* permite visualizar todos os objetos de ambos os bancos através das janelas *Master Database* e *Target Database*. Na janela *Script View* são listados todos os *scripts* necessários para realizar a sincronização entre o banco de dados *Master* e *Target*. Veja a interface do *DB Comparer* após a comparação na **Figura 3**.

Você pode alterar a direção da sincronização através do campo *Direction of DDL synchronization* disponível na barra de botões da janela principal da ferramenta. Dependendo da direção selecionada a lista de *scripts* é alterada para realizar a sincronização correta dos objetos. A ferramenta permite navegar entre os objetos tanto pela lista de *scripts* quanto pelo *treeview* dos objetos.

Selecionando um objeto no *treeview* da janela *Master*, por exemplo, o mesmo objeto é selecionado na janela *Target*, caso exista, e a instrução DDL dos mesmos é mostrada nas janelas *Master Object Definition* e *Target Object Definition*. Você pode ainda realizar filtros nos objetos do *treeview* a partir da opção *View* disponível na barra de botões da janela principal.

Objetos idênticos são mostrados no *treeview* na cor preta precedidos por um sinal de igual (=) enquanto que objetos alterados são mostrados em azul precedidos por um sinal de diferente (≠). Objetos inexistentes aparecem desabilitados no *treeview* precedidos por um sinal de menos (-) enquanto que novos objetos aparecem em verde, precedidos por um sinal de mais (+).

Ao selecionar um objeto alterado, inexistente ou adicionado, os *scripts* gerados referente ao mesmo para a sincronização são selecionados na aba *Script View* no lado direito da janela principal.

Nota: Vale ressaltar que o *DB Comparer* é destinado exclusivamente a comparação de estruturas de bancos de dados IB/FB, não realizando comparações e nem sincronizações dos dados armazenados nas tabelas.

O *DB Comparer* permite editar os *scripts* gerados durante a comparação, para isso basta realizar um duplo clique no *script* a ser alterado dentro da lista. A janela *SQL Script Editor* é aberta para que possamos editar o *script*. Através das teclas CTRL+SPACE podemos invocar o *Code Completion* do *DB Comparer*, semelhante ao Delphi (**Figura 4**).

O *DB Comparer* possui alguns *templates* pré-definidos que podem ser utilizados dentro do *SQL Script Editor*. Através do menu *Options>Keyboard Templates* temos acesso a janela *Keyboard Templates*. Nela, podemos visualizar todos os *templates* disponíveis e ainda criar nossos próprios *templates*.

Para utilizá-los dentro do *SQL Script Editor* basta digitar os caracteres de atalho e clicar na tecla espaço. Por exemplo, para adicionar comentários ao script digite "//" e tecle espaço, aparecerá no editor os caracteres /* */ para que o comentário possa ser escrito.

Você pode ainda visualizar o resultado da comparação dos bancos de dados através da aba *Table View* que permite a visualização na forma de Grid e habilita ainda a aplicação de filtros e agrupamentos. Para carregar a *Table View* selecione-a e através do menu de contexto, execute a opção *Fill Table View*. A partir do menu de contexto da janela *Table View* podemos exportar as informações para diversos formatos, como: Excel, HTML, XML e TXT.

Após verificar as diferenças e editar os *scripts* gerados, caso necessário, basta clicar na opção *Execute All Checked Scripts* disponível na barra de botões para executar os *scripts* DDL e realizar a sincronização entre os bancos de dados.

A execução dos *scripts* aparece dentro da janela *SQL Script Editor* e caso algum erro ocorra durante a execução o mesmo é mostrado ao usuário e o *script* é interrompido.

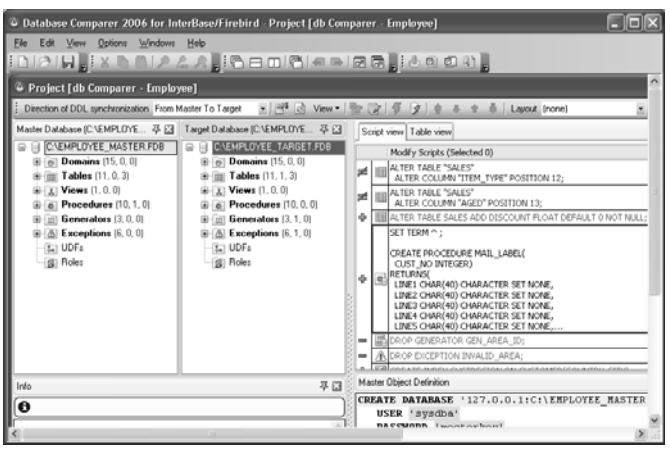


Figura 3. Diferenças encontradas nos bancos de dados Master e Target

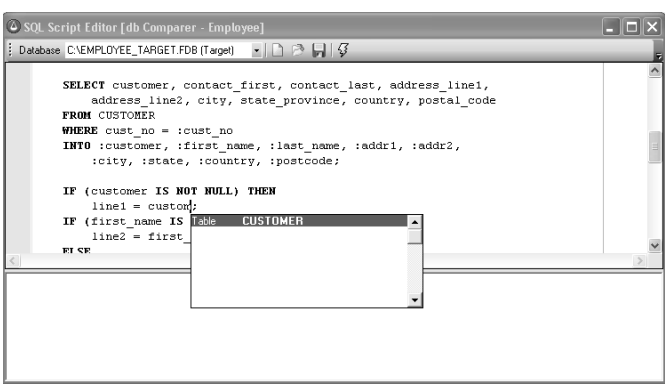


Figura 4. Alterando ou criando scripts DDL através do SQL Script Editor

Dica: Após a execução de um ou mais *scripts* DDL, basta usar a tecla F5 *(Refresh)* para realizar uma nova comparação dos bancos e atualizar as listas de objetos.

O *DB Comparer* permite através de três janelas de opções customizarmos diversas características de seu IDE. Através da opção *Options>Environment Options* disponível no menu principal, por exemplo, podemos alterar diversas opções do IDE, como opções de comparação, confirmação, cores, fontes, entre outros.

Através da opção *Options>Editor Options*, podemos customizar a janela *SQL Script Editor* assim como muitas de suas funcionalidades. Por último temos a janela *Visual Options* que pode ser acessada através do menu *Options>Visual Options*. Nessa janela podemos selecionar diversos *Look and Feel* para serem aplicados ao IDE.

Várias opções permitem alterar a forma de visualização de diversos componentes do IDE, como botões, abas, *checkbox*, barra de menus e botões, entre outros. Uma vez definida as configurações de visualização dos objetos do IDE, temos a opções de salvar as mesmas criando assim um novo tema (*Look and Feel*).

Relatórios

A ferramenta permite a criação e emissão de relatórios a partir das informações contidas na aba *Table View*. Após executar os filtros e agrupamentos necessários nas informações referente as diferenças encontradas na comparação, você pode criar relatórios para visualizar e imprimir tais dados. Para acessar as funcionalidades de relatórios do *DB Comparer* selecione a opção *View>Reports* disponível no menu principal. Dentro da janela *Reports* podemos criar, alterar e visualizar relatórios do projeto.

Por padrão a ferramenta traz o relatório *Object Properties* que mostra a instrução DDL e os valores das propriedades de

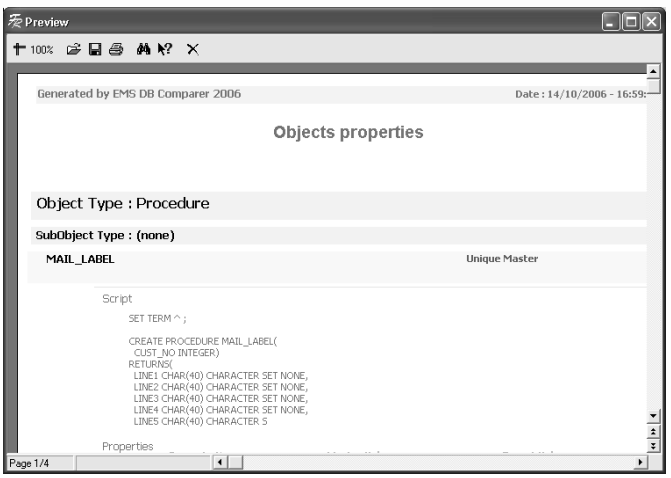


Figura 5. Relatório Objects Properties disponível no DB Comparer

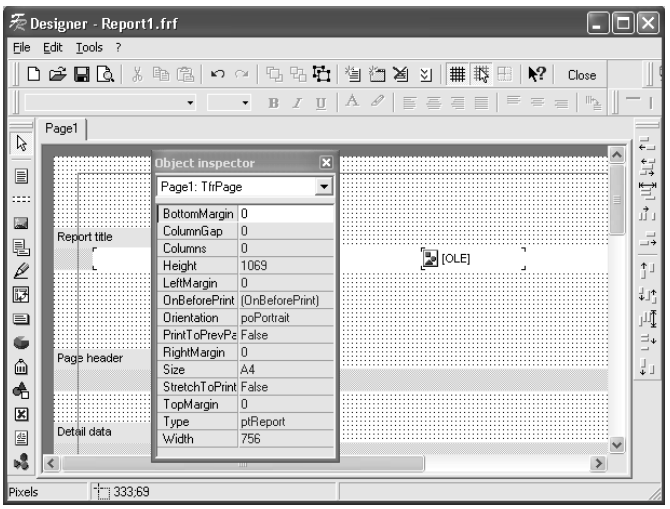


Figura 6. Editor do FastReport para criação e edição de relatórios no DB Comparer

cada objeto presente no *Table View*. Antes de executar o relatório, certifique apenas de existir informações na aba *Table View* para serem visualizadas.

Selecione o relatório *Object Properties* e clique no botão *View*. O *DB Comparer* possui um preview para visualização e impressão de seus relatórios (**Figura 5**).

Através da janela de preview podemos salvar o relatório (formato FRP) através do botão *Save Report* ou ainda imprimí-lo utilizando o botão *Print Report*. Caso haja a necessidade de alterar a estrutura do relatório, você pode fazê-lo através da opção *Edit* disponível na barra de botões da janela *Reports*.

O *DB Comparer* utiliza o *FastReport* (*www.fast-report.com*) para edição e visualização de relatórios. Os desenvolvedores que utilizam o *FastReport* em seus aplicativos Delphi ou C++ Builder não terão dificuldades em criar e alterar relatórios. O *FastReport* permite a utilização de recursos equivalentes ao QuickReport como criação de relatórios baseados em páginas e bandas.

Você pode utilizar imagens, agrupamentos, gráficos, sub-relatórios, objetos OLE, códigos de barra, e até mesmo formulários dentro da ferramenta. Para criar um relatório utilize a opção *Create* disponível na janela *Reports* (**Figura 6**).

Aplicação Console

Como comentado anteriormente, o *DB Comparer* possui uma versão console, que pode ser executada através de linha de comando em uma janela *prompt* do DOS. Você pode acessar a ferramenta através do arquivo *IbComparerc.exe* disponível no diretório de instalação do produto.

A execução da aplicação é bastante simples, basta abrir uma janela do DOS e digitar o seguinte comando:

```
IbComparerc.exe C:\DBComparer-Employee.icp scripts.sql
```

O primeiro parâmetro, *C:\DBComparer-Employee.icp*, diz respeito ao caminho completo do arquivo de projeto a ser executado, para o segundo parâmetro devemos informar o nome do arquivo a ser gerado que conterá os *scripts* resultantes da comparação entre os bancos de dados configurados no projeto.

Você pode ainda incluir um terceiro parâmetro não obrigatório, "/E", que faz com que os *scripts* gerados a partir da comparação sejam executados no final do processo.

Conclusão

Conhecemos neste artigo as principais características e funcionalidades do *DB Comparer*, ferramenta que pode ser bastante útil em processos de comparação de estruturas de bancos de dados InterBase e Firebird. Um grande abraço e até a próxima. ■

ClubeDelphi PLUS

Acesse agora mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Everson Volaco mais dicas sobre a ferramenta DB Comparer.

www.devmedia.com.br/articles/viewcomp.asp?comp=3302

Exibindo dicas na forma de balão com a descrição de campos do FB

RODRIGO LAZOTI
(rodrigolazoti@yahoo.com.br)
é programador e desenvolvedor Delphi, .NET, Java e ASP. Trabalha atualmente com Delphi/Firebird em conjunto com dbExpress em projetos cliente/servidor e multicamadas.

Imagine um formulário com vários campos espalhados e você tendo que explicar a finalidade de cada campo para o usuário, parece uma tarefa um pouco chata, não acha? Agora imagine o usuário posicionando o cursor do mouse sobre o campo e automaticamente surgir um *hint* com formato de um balão descrevendo a finalidade do campo. Sim, isso é possível e mostrarei como fazer essa funcionalidade neste artigo.

Criaremos uma tabela em um banco de dados Firebird (nada impede que seja utilizado outro SGBD) com alguns campos e incluir descrições para os mesmos. Após concluir essa etapa, criaremos uma rotina no Delphi para resgatar essas descrições e associar automaticamente com o seu respectivo *DBEdit* contido no formulário para manipulação dos dados da tabela.

Como resgatar as descrições dos campos no Firebird

O Firebird armazena diversas informações sobre o banco de dados em tabelas de sistema, em nosso exemplo utilizaremos a tabela RDB$RELATION_FIELDS, que mantém uma lista dos campos das tabelas e as informações das características de uma coluna.

A idéia aqui não é criar uma coluna de descrição (que poderia ser adaptada facilmente para este exemplo), mas sim usar a descrição armazenada pelo próprio Firebird. Os campos da tabela de sistema RDB$RELATION_FIELDS que nos interessam estão descritos na **Tabela 1**.

Campo	Descrição
RDB$RELATION_NAME	Contém o nome da tabela do banco de dados
RDB$FIELD_NAME	Contém o nome do campo da tabela
RDB$DESCRIPTION	Contém a descrição do campo da tabela

Tabela 1. Campos da tabela RDB$RELATION_FIELDS que serão usados no exemplo

Criando o banco de dados e a tabela de exemplo

O primeiro passo é criar o banco, e para facilitar a manipulação dos objetos, utilizarei o IBExpert da HK-Software. Ele possui uma versão *freeware* que pode ser obtida no site *www.ibexpert.com*. Este artigo não visa explicar como criar um banco ou até mesmo tabelas no Firebird.

Crie um banco de dados, salve-o com o nome de "BancoDados.fdb" e insira uma tabela chamada "CLIENTES". Os campos da tabela CLIENTES são mostrados na **Tabela 2**.

Campo	Tipo	Tamanho	Descrição
CODIGO	Integer	–	Código do cliente
NOME	Varchar	50	Nome do cliente
CPF	Varchar	11	CPF do cliente (Informe somente números, sem ponto e traços)

Tabela 2. Campos da tabela CLIENTES que será usada em nosso exemplo

Veja na **Figura 1** como ficou a criação da tabela e suas respectivas colunas.

Criando o aplicativo

Agora que temos o banco de dados pronto para o exemplo, vamos criar o aplicativo no Delphi. Inicie o Delphi e mude a propriedade *Name* do formulário para "frmPrincipal", salve a unit desse formulário com o nome de "uPrincipal.pas". Adicione um *SQLConnection* e dois *SQLDatasets* (aba *dbExpress*). Configure suas propriedades conforme a **Tabela 3**.

Teste a conexão com o banco de dados mudando a propriedade *Connected* do *SQLConnection1* para *True*, caso ocorra algum erro, reveja os valores das propriedades. Se conectou corretamente mude novamente o valor da propriedade *Connected* para *False*.

Dê um duplo clique no *SQLDataset1*, será aberta uma janela dos campos do componente, clique com o botão direito do mouse e escolha a opção *Add all fields*, para que sejam adicionados todos os campos da tabela. Repita essa operação no *SQLDataset2*.

Continuando, através da aba *Data Access* insira um *DataSetProvider*, um *ClientDataset* e um *DataSource* ao formulário e

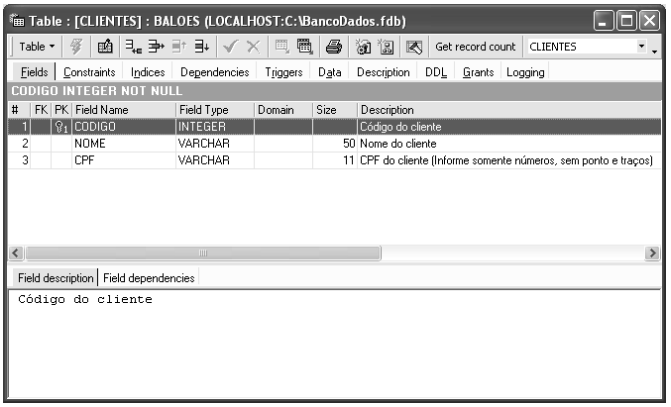


Figura 1. Tabela CLIENTES que será usada em nosso exemplo

configure suas propriedades conforme a **Tabela 4.**

Componente	Propriedade	Valor
DataSetProvider1	DataSet	SQLDataSet1
ClientDataSet1	ProviderName	DataSetProvider1
DataSource1	DataSet	ClientDataSet1

Tabela 4. Configuração dos componentes

Repita a mesma operação para adicionar todos os campos, feita nos componentes *SQLDataset1* e *SQLDataset2*, mas para o *ClientDataset1*. Em seguida, ainda na janela de campos do *ClientDataset1*, selecione todos os campos, arraste e solte-os no formulário, assim será criado automaticamente todos os *DBEdits* para seus respectivos campos do *ClientDataset*. Nessa etapa seu formulário deve estar parecido com a **Figura 2**.

ClubeDelphi PLUS

Acesse agora mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como instalar e trabalhar com o IBExpert.

www.devmedia.com.br/articles/viewcomp.asp?comp=3082

Componente	Propriedade	Valor
SQLConnection1	ConnectionName	IBConnection
SQLConnection1	LoginPrompt	False
SQLConnection1	Params [Database]	localhost:<caminho>\BancoDados.fdb
SQLConnection1	Params [SQLDialect]	3
SQLConnection1	VendorLib	fbclient.dll
SQLDataset1	SQLConnection	SQLConnection1
SQLDataset1	CommandText	select * from CLIENTES
SQLDataset2	SQLConnection	SQLConnection1
SQLDataset2	CommandText	select RDB$FIELD_NAME as CAMPO, RDB$DESCRIPTION as DESCRICAO from RDB$RELATION_FIELDS where RDB$RELATION_NAME = :Tabela

Tabela 3. Configuração dos componentes dbExpress

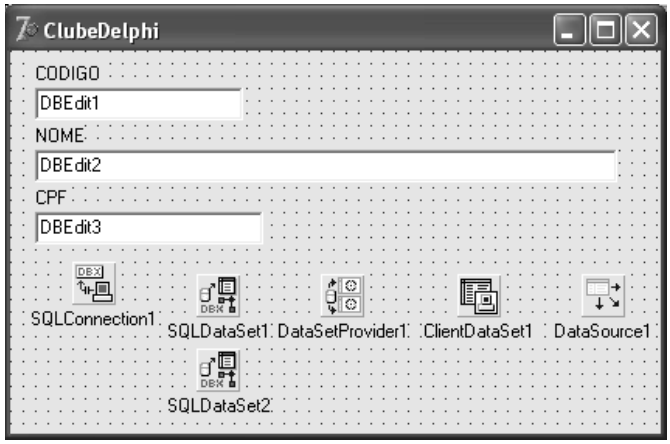
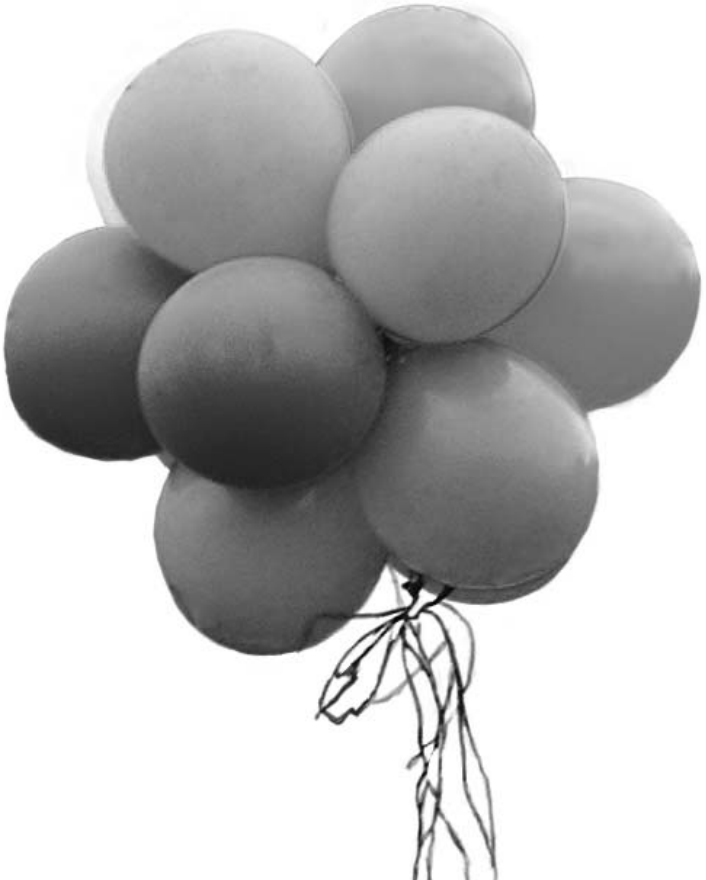


Figura 2. Formulário principal do exemplo

> **Nota:** Para esse exemplo simples, não usaremos um Data Module, que é aconselhável em aplicações reais.

Criando a rotina para exibir os balões

Chegamos na parte mais interessante do artigo, vamos codificar nossa rotina. O primeiro passo será incluir na sessão *uses* as units *ToolWin* e *Commctrl*. Vamos criar todas as variáveis e constantes que serão usadas pelas rotinas do exemplo. Na **Listagem 1** temos as variáveis e constantes que devem ser criadas, a variável *frmPrincipal* já foi criada automaticamente pelo Delphi.



istagem 1. Declarando as variáveis e constantes

```
var
  frmPrincipal: TfrmPrincipal;
  hTooltip: Cardinal;
  ti: TToolInfo;

const
  TITULO = 'Clube Delphi';
  TTS_BALLOON = $40;
  TTM_SETTITLE = (WM_USER + 32);
```

Crie dois métodos na sessão *public*, conforme mostra o código a seguir, e aperte CTRL+SHIFT+C para que sejam criados os corpos dos métodos automaticamente.

```
public
  procedure InsereTextoBalao(Objeto: TWinControl;
    NomeCampo: string);
  procedure addBaloes(NomeTabela: string);
end;
```

InsereTextoBalao tem a finalidade de verificar se o campo existe (parâmetro *NomeCampo*), se existir, verifica se tem uma descrição para esse campo, caso haja, a *procedure* associa a descrição com o balão do *DBEdit* (parâmetro *Objeto*).

AddBaloes será o método para realizar a inclusão dos balões nos campos do formulário, só temos que passar a tabela (parâmetro *NomeTabela*) que contém os campos para serem associados com os *DBEdtits* do formulário.

Uma terceira *procedure* será adicionada ao projeto, porém essa não fará parte do formulário, mesmo estando em sua unit, a *procedure* se chama *AdicionaBalao* e tem como finalidade criar o *hint* no formato de balão e adicionar o ícone de informação ao balão. Ela não precisa ser declarada na sessão *public* do formulário. Na **Listagem 2** temos o código das três *procedures*.

Pronto, já temos a rotina codificada, agora basta usá-la no *OnCreate* do formulário e junto fazer a conexão com o Firebird, conforme o seguinte código:

```
procedure TfrmPrincipal.FormCreate(Sender: TObject);
begin
  SQLConnection1.Connected := True;
  addBaloes('CLIENTES');
end;
```

Nosso exemplo já está pronto, agora é só executá-lo e posicionar o cursor do mouse sob os campos, para ver o resultado (**Figura 3**).

Conclusão

Como vimos nesse artigo, essa é uma forma simples de melhorar o visual de uma aplicação e torná-la mais intuitiva para o usuário final.

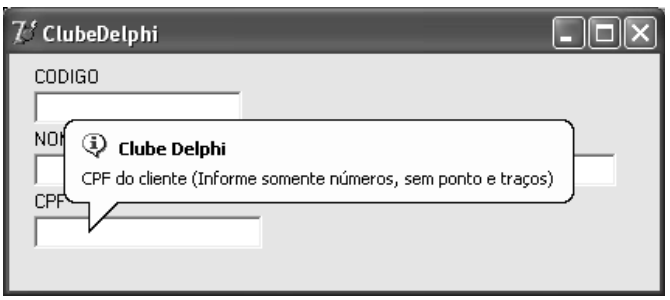


Figura 3. Executando o exemplo

Listagem 2. Procedimentos do projeto

```
procedure AdicionaBalao(hwnd: dword; lpti: PToolInfo;
  IconType: Integer; Text, Title: PChar);
var
  Item: THandle;
  Rect: TRect;
  buffer: array[0..255] of char;
begin
  Item := hWnd;
  if (Item<>0) and (GetClientRect(Item, Rect)) then
  begin
    lpti.hwnd := Item;
    lpti.Rect := Rect;
    lpti.lpszText := Text;
    SendMessage(hToolTip, TTM_ADDTOOL, 0,
      Integer(lpti));
    FillChar(buffer, sizeof(buffer), #0);
    lstrcpy(buffer, Title);
    if (IconType>3) or (IconType<0) then
      IconType := 0;
    SendMessage(hToolTip, TTM_SETTITLE, IconType,
      Integer(@buffer));
  end;
end;

procedure TfrmPrincipal.InsereTextoBalao(
  Objeto: TWinControl; NomeCampo: string);
var
  foiCriado: Boolean;
begin
  if (Assigned(Objeto)) then
    if (SQLDataSet2.Locate('CAMPO', NomeCampo,
      [locaseinsensitive, loPartialKey])) then
      if (SQLDataSet2.FieldByName(
        'DESCRICAO').AsString <> EmptyStr) then
      begin
        if not (foiCriado) then
        begin
          hToolTip := CreateWindowEx(0,
            'Tooltips_Class32', nil,
            TTS_ALWAYSTIP or TTS_BALLOON,
            Integer(CW_USEDEFAULT),
            Integer(CW_USEDEFAULT),
            Integer(CW_USEDEFAULT),
            Integer(CW_USEDEFAULT), Handle, 0,
            hInstance, nil);
          if (hToolTip<>0) then
          begin
            SetWindowPos(hToolTip, HWND_DESKTOP, 0, 0,
              0, 0, SWP_NOMOVE or SWP_NOSIZE or
              SWP_NOACTIVATE);
            ti.cbSize := SizeOf(TToolInfo);
            ti.uFlags := TTF_SUBCLASS;
            ti.hInst := hInstance;
            BringWindowToTop(hTooltip);
          end;
          foiCriado := True;
        end;
        AdicionaBalao(Objeto.Handle, @ti, 1, pChar(
          SQLDataSet2.FieldByName(
          'DESCRICAO').AsString), pChar(TITULO));
      end;
end;

procedure TfrmPrincipal.addBaloes(NomeTabela: string);
var
  i: Integer;
begin
  try
    SQLDataSet2.ParamByName('Tabela').AsString :=
      NomeTabela;
    SQLDataSet2.Open;

    if not (SQLDataSet2.IsEmpty) then
    begin
      for i := 0 to ComponentCount-1 do
      begin
        if (Components[i].ClassNameIs('TDBEdit')) then
          InsereTextoBalao(TWinControl(Components[i]),
            TDBEdit(Components[i]).DataField);
      end;
    end;
    SQLDataSet2.Close;
  except
    ShowMessage('Erro ao criar balão.');
  end;
end;
```

Criando um menu recursivo

GABRIEL F. JANK
(gabrieljank@gmail.com)
Delphiano há mais de cinco anos, com experiência nas áreas de Web Services, Sistemas Distribuídos e Orientação a Objetos. Ex-aluno do Curso Técnico em Informática do Colégio Frederico Jorge Logemann de Horizontina/RS e Bacharelando em Sistemas de Informações pela SETREM de Três de Maio/RS.

Atualmente, os sites de comércio eletrônico são muito comuns e presentes em nosso cotidiano. Com o passar dos tempos mais e mais pessoas estão começando a comercializar através dos meios eletrônicos, isso graças às tecnologias de segurança da informação que trazem cada vez mais confiança as operações.

Falando de base tecnológica, o Delphi traz o ASP.NET como uma das opções mais emergentes no cenário de desenvolvimento para a Web. O intuito deste artigo é demonstrar a utilização do ASP.NET no Delphi em um exemplo simples que visa demonstrar a criação de um menu dinâmico, com a estrutura de departamentos e produtos, semelhante às estruturas existentes hoje nos principais sites voltados ao e-commerce.

Para isso vamos utilizar o Firebird 1.5, com acesso a dados através do *Firebird Data Provider*, que é um *Provider* ADO.NET específico para o Firebird, desenvolvido e distribuído no mesmo site do banco de dados, que deve ser instalado em nosso ambiente de desenvolvimento que será o Delphi 2006 (na edição 66 da ClubeDelphi temos um artigo que mostra como instalar o *Provider* do Firebird).

Em termos de desenvolvimento, vamos demonstrar também a utilização de métodos recursivos que é uma opção inteligente para resolver problemas computacionais, digamos assim, com estilo.

Problema e escopo

Hoje não falamos e não pensamos em sites estáticos, desenvolvidos puramente em HTML, voltamos para ferramentas e tecnologias dinâmicas, que utilizam bases relacionais para o armazenamento dos dados, dados esses que serão responsáveis pela "geração" de nossas páginas.

Em se tratando do e-commerce, um dos problemas encontrados na maioria dos casos é referente à classificação e disposição dos produtos na Web. Existe uma série de departamen-

tos, que possuem vários produtos relacionados. Surge então o problema: Como podemos desenvolver uma estrutura dinâmica e simples para os departamentos e produtos em um site de comércio eletrônico?

A dinamicidade faz com que as informações referentes aos departamentos e produtos venham de um banco de dados, e que a manutenção desses também reflita automaticamente na estrutura e disposição do nosso site.

Este artigo refere-se simplesmente a disposição de produtos em uma série de departamentos definidos, questões como pedidos, carrinho de compras, sessões e outros desenvolvimentos fundamentais para o funcionamento do comércio eletrônico estão fora do escopo (para saber mais sobre comércio eletrônico, veja o mini-curso de uma aplicação Web completa, iniciado na edição 74 da ClubeDelphi).

Analisando o cenário

Vamos dar início à atividade, e o primeiro passo é pensarmos na estrutura do banco de dados. Temos teoricamente focadas duas entidades em nosso problema: *Departamentos* e *Produtos*, além disso, existe uma relação onde um *Produto* estará alocado a um determinado *Departamento*, e outra premissa onde um departamento pode referenciar-se a outro departamento tornando-se assim uma subcategoria de produtos.

Segue o DER (*Diagrama de Entidades e Relacionamentos*) na **Figura 1**.

Ainda comentando sobre o DER, temos então a entidade de *Departamentos* que possui uma auto-relação que identificará as subcategorias existentes, formando assim uma "árvore" com todos os departamentos da base.

Existe ainda uma relação entre os produtos e os departamentos, onde teremos para cada departamento uma série de produtos associados. Para finalizar temos uma *View* que servirá para montar o menu de departamentos e subcategorias, além de trazer a informação referente ao número de produtos associados.

Desenvolvendo o projeto

Com base na estrutura apresentada, podemos iniciar o desenvolvimento do projeto. O primeiro passo é, a partir do DER, criar o banco de dados. Na **Listagem 1** temos o *script* completo do banco para Firebird 1.5, onde demos o nome de "ECOMMERCE.FDB".

Listagem 1. Script do banco de dados

```
SET SQL DIALECT 3;
SET NAMES NONE;
CREATE DATABASE <caminho>'\ECOMMERCE.fdb'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 4096
DEFAULT CHARACTER SET NONE;

CREATE TABLE DEPARTAMENTOS (
  ID          INTEGER NOT NULL,
  REF         INTEGER,
  DESCRICAO   VARCHAR(50)
);

CREATE TABLE PRODUTOS (
  CODPRODUTO      INTEGER NOT NULL,
  DEPARTAMENTO    INTEGER NOT NULL,
  DESCPRODUTO     VARCHAR(60),
```

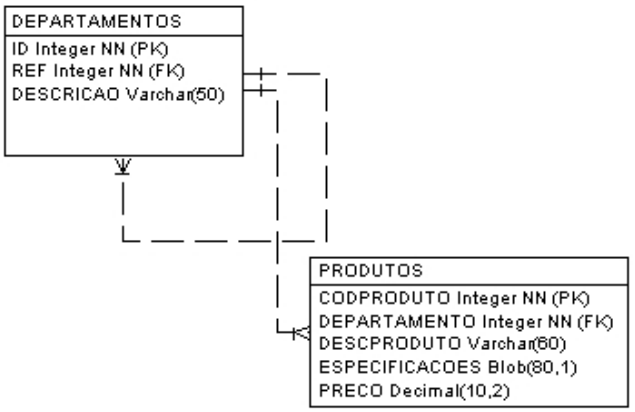


Figura 1. Diagrama de Entidades e Relacionamentos

```
  ESPECIFICACOES  BLOB SUB_TYPE 1 SEGMENT SIZE 80,
  PRECO           DECIMAL(10,2)
);

CREATE VIEW PROD_DPTOS(
  REF, DESCRICAO, ID, NUMERO)
AS
SELECT D.REF, D.DESCRICAO, D.ID, COUNT(P.CODPRODUTO)
FROM DEPARTAMENTOS D
LEFT OUTER JOIN PRODUTOS P ON (P.DEPARTAMENTO = D.ID)
GROUP BY D.REF, D.DESCRICAO, D.ID
;

ALTER TABLE DEPARTAMENTOS ADD PRIMARY KEY(ID);
ALTER TABLE PRODUTOS ADD PRIMARY KEY (CODPRODUTO);
ALTER TABLE DEPARTAMENTOS ADD FOREIGN KEY (REF)
REFERENCES DEPARTAMENTOS (ID) ON DELETE NO ACTION
ON UPDATE NO ACTION;
ALTER TABLE PRODUTOS ADD FOREIGN KEY (DEPARTAMENTO)
REFERENCES DEPARTAMENTOS (ID) ON DELETE NO ACTION
ON UPDATE NO ACTION;
```

Na **Listagem 2** temos o *script* com alguns dados para preenchimento das tabelas.

Listagem 2. Script de dados para preenchimento das tabelas

```
INSERT INTO DEPARTAMENTOS (ID, REF, DESCRICAO) VALUES
  (1, NULL, 'Departamentos');
INSERT INTO DEPARTAMENTOS (ID, REF, DESCRICAO) VALUES
  (2, 1, 'Livros');
INSERT INTO DEPARTAMENTOS (ID, REF, DESCRICAO) VALUES
  (3, 1, 'Revistas');
INSERT INTO DEPARTAMENTOS (ID, REF, DESCRICAO) VALUES
  (4, 1, 'CD-Roms');
INSERT INTO DEPARTAMENTOS (ID, REF, DESCRICAO) VALUES
  (5, 2, 'Técnicos');
INSERT INTO DEPARTAMENTOS (ID, REF, DESCRICAO) VALUES
  (6, 2, 'Literatura');
INSERT INTO DEPARTAMENTOS (ID, REF, DESCRICAO) VALUES
  (7, 2, 'Romances');
INSERT INTO DEPARTAMENTOS (ID, REF, DESCRICAO) VALUES
  (8, 3, 'Moda');
INSERT INTO DEPARTAMENTOS (ID, REF, DESCRICAO) VALUES
  (9, 3, 'Culinária & Gastronomia');
INSERT INTO DEPARTAMENTOS (ID, REF, DESCRICAO) VALUES
  (10, 3, 'Ciências');
INSERT INTO DEPARTAMENTOS (ID, REF, DESCRICAO) VALUES
  (11, 4, 'Programas e Utilitários');
INSERT INTO PRODUTOS (CODPRODUTO, DEPARTAMENTO,
  DESCPRODUTO, PRECO) VALUES
  (1, 5, 'Delphi 7: A bíblia', 79);
INSERT INTO PRODUTOS (CODPRODUTO, DEPARTAMENTO,
  DESCPRODUTO, PRECO) VALUES
  (2, 10, 'Galileu', 9);
COMMIT WORK;
```

Vamos iniciar agora o desenvolvimento do aplicativo Web. Para criar uma nova aplicação ASP.NET no Delphi, acesse o

menu *File>New>ASP.NET Web Application - Delphi for .NET*.

Dê o nome de "LojaVirtual" ao projeto. Renomeie a página principal de *WebForm1.aspx* para "Index.aspx", para que o servidor web saiba que ela se trata da página principal do site.

Como citado anteriormente, vamos utilizar o *Firebird Data Provider* para o acesso ao banco, então vamos adicionar ao formulário o componente de conexão a dados: o *FbConnection*. Dê o nome de "fbCon" e acesse a propriedade *Connection String* para abrir o editor de conexão do componente.

No editor, informe o caminho do banco de dados, por exemplo: localhost:C:\ECOMMERCE.FDB na opção *Database*. As credenciais de acesso em *User* e *Password* (padrão SYSDBA e *masterkey*) também devem ser preenchidas. Observe que temos mais opções para configuração, como o *Charset*, porta, *dialect* etc. Pode-se utilizar o botão *Test* para testar a conexão com a base (**Figura 2**).

Além da conexão, vamos utilizar mais uma série de componentes para montar nossa consulta de produtos. Adicione um *FbDataAdapter* dando o nome de "fbDAProdutos". Um *wizard* será aberto e a primeira informação é a conexão: *fbCon*.

> **Nota:** Se o *wizard* não for mostrado, acesse-o através do menu de contexto, escolhendo a opção *Configure Data Adapter*.

Na próxima tela devemos informar que vamos utilizar comandos SQL, então deixe marcado a opção *Use SQL statements*. O comando SQL será:

```
SELECT DEPARTAMENTO, DESCPRODUTO, PRECO
FROM PRODUTOS
WHERE DEPARTAMENTO = @DPTO
```

Finalizamos assim, a configuração. Após isso, clique na propriedade *SelectCommand>Parameters* do *fbDAProdutos* e vamos adicionar o parâmetro *@DPTO*, conforme mostrado na **Figura 3**.

Vamos montar o layout da página, que será bem básico, contendo somente um Grid com os produtos e o menu que será montado dinamicamente pela aplicação através de um método recursivo.

Adicione então um *DataGrid* e dê o nome de "DGProdutos". Altere a propriedade *AutoGenerateColumns* para *False*. Clique em *Property Builder* e vamos configurar os campos (*Columns*). Adicione duas colunas digitando "Produto" e "Preço" em *Header Text*.

Para *Data Field*, digite DESCPRODUTO e PRECO. Para o campo *Preço*, adicione "{0:c}" no item *Data formatting expression*. Veja como ficou o *layout* final da página na **Figura 4**.

Método Recursivo

Método ou função que durante o seu processamento possui uma nova chamada para si mesmo (a), desencadeando assim uma recursividade. Os métodos recursivos devem ser bem pensados, pois novas chamadas podem gerar uma laço infinito de processamento.

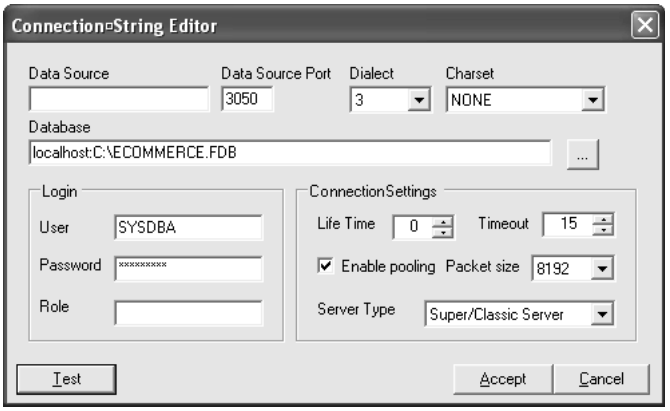


Figura 2. Editor de conexão do FbConnection

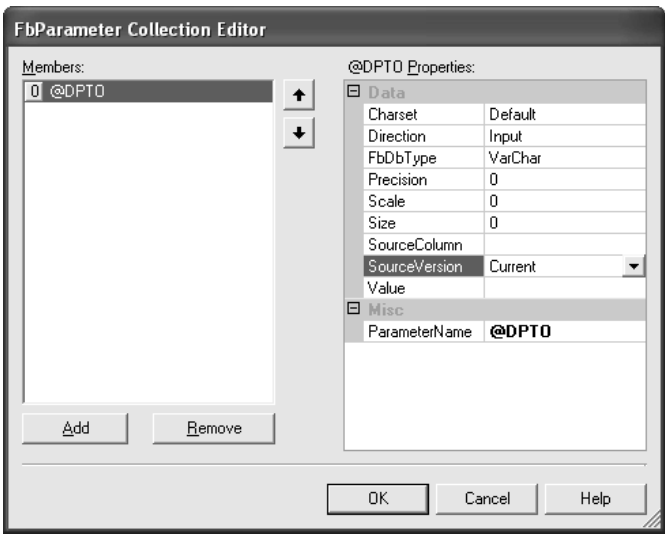


Figura 3. Adicionando parâmetros ao FbDataAdapter

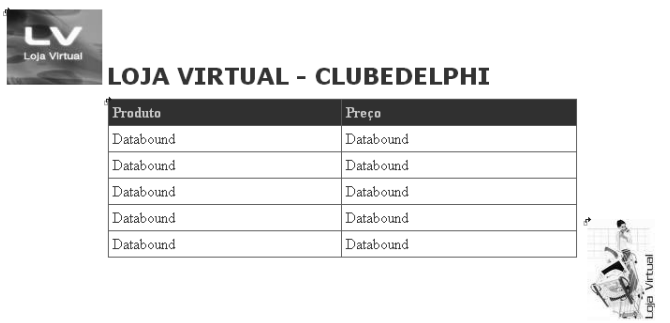


Figura 4. Layout da página em desenvolvimento

Com base na *view Estrutura*, podemos desenvolver um método recursivo que faça a criação de todo menu de departamentos do site, já que temos vários departamentos inter-relacionados.

Na **Listagem 3** temos o código do método *Menu* que será declarado na seção *public* da página.

Listagem 3. Método Recursivo Menu

```
procedure TWebForm1.Menu(ID: string);
var
```

```
  sql: FbCommand;
  reader: FbDataReader;
  Link: HyperLink;
  texto: HtmlGenericControl;
  i: integer;
begin
  sql := FbCommand.Create;
  sql.Connection := FbCon;
  { Verifica se o menu está sendo iniciado }
  if ID = '' then
  begin
    { Item inicial da árvore }
    sql.CommandText :=
      'SELECT p.id, p.descricao, p.ref, p.numero '+
      'FROM prod_dptos p '+
      'where p.ref is null';
  end
  else
  begin
    { Verificação recursiva }
    sql.CommandText :=
      'SELECT p.id, p.descricao, p.ref, p.numero '+
      'FROM prod_dptos p '+
      'where p.ref = @REF';
    sql.Parameters.Add('@REF', FbDbType.VarChar);
    sql.Parameters[0].Value := ID;
  end;
  sql.Prepare;
  { Será criada uma instância de dados para cada
    nível do menu }
  Reader := sql.ExecuteReader();
  if (Reader <> nil) then
  begin
    { Laço do menu }
    while (Reader.Read()) do
    begin
      { Cria a variável para um novo link HTML }
      Link := HyperLink.Create;
      texto := HtmlGenericControl.Create;
      { Adiciona espaços para edentar o menu }
      for i := 0 to nivel do
        texto.InnerHtml := texto.InnerHtml + ' ';
      Self.Controls.Add(texto);
      { Se o departamento não possui produtos
        vinculados }
      if Reader.GetInt32(3) = 0 then
      begin
        Link.Text := Reader.GetString(1);
        Link.Font.Bold := True;
      end
      else
      begin
        Link.NavigateUrl := './Index.aspx?DPTO='+
          Convert.ToString(Reader.GetInt32(0));
        Link.Text := Reader.GetString(1)+' ('+
          Convert.ToString(Reader.GetInt32(3))+')';
        Link.Font.Bold := False;
      end;
      { Propriedades de fonte do link }
      Link.Font.Size := 10-nivel;
      Link.Font.Name := 'Verdana';
      Link.Page := Self;

      { O item inicial do menu não será exibido }
      if ID <> '' then
        Self.Controls.Add(Link);

      { Adiciona uma quebra entre os links }
      texto := HtmlGenericControl.Create;
      texto.InnerHtml := '<br>';
      Self.Controls.Add(texto);

      { Incrementa o nível }
      inc(nivel);
      { Recursividade: Chamo novamente Menu }
      Menu( Convert.ToString(Reader.GetInt32(0)) );
      { Decrementa um nível: fim dos registros
        de cada nível }
      dec(nivel);
    end;
    Reader.Close();
  end;
end;
```

Uma observação importante, é que devemos declarar a variável *nivel* na seção *public* dentro da estrutura da classe *TWebForm*, pois necessitamos de uma variável global:

```
TWebForm1 = class(System.Web.UI.Page)
...
  public
    nivel: integer;
...
```

Todos os itens chamam o mesmo método *Menu* que permite a abertura da árvore em um número "x" de níveis, menos os departamentos com produtos vinculados, esses exibirão um contador com o totalizador ao lado e a chamada resultará em uma grade com a lista de produtos de tal departamento. Na **Listagem 4** temos o código que deve ser incluído no evento *Load* da página.

Listagem 4. Evento Load da página Index.aspx

```
procedure TWebForm1.Page_Load(sender: System.Object;
  e: System.EventArgs);
begin
  fbCon.Open();
  try
    FbDAProdutos.SelectCommand.Parameters[
      '@DPTO'].Value :=
    Request.QueryString.Item['DPTO'];
    DGProdutos.DataSource :=
      FbDAProdutos.SelectCommand.ExecuteReader();
    DGProdutos.DataBind();
    Menu('');
  finally
    fbCon.Close();
  end;
end;
```

O resultado do menu e da grade de itens pode ser visualizado na **Figura 5**.

Conclusão

Podemos concluir que o ASP.NET é uma tecnologia madura para o desenvolvimento de aplicações voltadas para a Internet. No caso, voltamos nossas atenções para o comércio eletrônico, além disso, podemos notar a utilidade das funções recursivas e como elas podem nos auxiliar na resolução de problemas relacionados ao desenvolvimento.

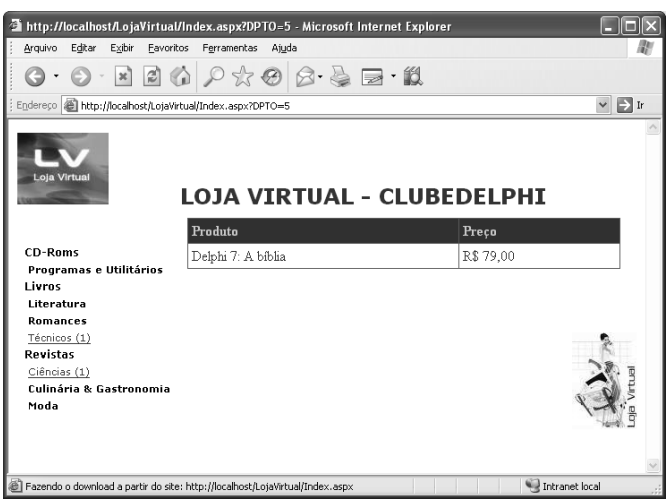


Figura 5. Menu e grade de produtos em funcionamento

Atualização automática de aplicações via FTP

FABIO CORRÊA
(fabio@progresystem.com.br)
é Diretor e analista de sistemas da ProgreSystem Informática. Trabalha com Delphi e Firebird, desenvolvendo aplicações cliente-servidor. Bacharel em Sistemas de informação pela Universidade do Sul de Santa Catarina - UNISUL na cidade de Tubarão-SC.

Quantas vezes nosso cliente possui uma versão desatualizada do sistema e que precisa ser substituída? Situação essa faz com que o pessoal do suporte técnico dispenda tempo na substituição do software. Num contexto em que a internet está amplamente disseminada, podemos transferir a responsabilidade de atualizar os sistemas para o cliente, fazendo com que custos com pessoal técnico capacitado sejam reduzidos.

A transferência dessa responsabilidade exige ter em mente que, quanto maior a simplicidade e automação dessa operação, mais satisfeito ficará o cliente. Assim neste artigo apresentamos uma solução baseada no protocolo FTP para atualizar um sistema.

O que é FTP? É um protocolo de transferência de arquivos baseado no TCP/IP. Enviar e receber arquivos é a principal função do FTP, sendo amplamente usado no upload de arquivos para servidores Web. Destaco duas qualidades inerentes a esse protocolo: simplicidade e rapidez na utilização.

Primeiros passos

Através do projeto, o usuário poderá atualizar o sistema numa forma simples e automatizada. Na tela principal vamos ter opções para configurar a conexão com o servidor FTP e a atualização propriamente dita.

Abra o Delphi 7 e vamos montar a tela principal, adicionando dois *Panels*, dois *Labels*, três *BitBtns*, um *Image*, um *Gauge*, um *StatusBar* e o *IdFTP* (não mencionei alguns *Labels* que servem para indicar as funcionalidades ao usuário). Altere as propriedades dos componentes conforme a **Tabela 1**.

Dê um duplo clique na *StatusBar* e adicione um *Panel*. Salve o projeto com o nome "update.dpr", a unit com o nome "unt_frmupdate.pas" e configure-os como mostra a **Figura 1**.

Agora vamos construir o módulo de configuração do funcionamento do FTP. Crie um formulário, alterando a propriedade

Name para "frmconfiguracao". Adicione seis *Edits* (dando os nomes de: "edftp", "edusuario", "edsenha", "edsalvar", "eddiretorio" e "edarquivo"), um *CheckBox* e um *BitBtn*.

No *BitBtn* altere a propriedade *Kind* para *bkOK* e *BorderStyle* do formulário para *bsDialog*. Em seguida distribua os componentes conforme indicado na **Figura 2**, salve a unit com o nome "unt_frmconfiguracao.pas". Os dados da configuração do FTP serão armazenados num arquivo INI.

Gravando o arquivo de configuração

Para manipular arquivos INI no Delphi é muito simples, primeiramente vou explicar a estrutura do arquivo INI, mostrado na **Listagem 1**.

```
DIRETORIOSERVIDORFTP=www
```

Observando a estrutura do arquivo INI, o valor [CONEXAO] corresponde ao nome da tabela. A seguir, temos o item FTP que corresponde ao campo da tabela, o mesmo recebe o valor *ftp.dominio.com.br*, e assim sucessivamente com os outros itens abaixo do FTP.

Para criar um arquivo INI, basta declarar a unit *IniFiles* no formulário da configuração. Veja o procedimento para criação do arquivo INI na **Listagem 2**.

> **Nota:** O nome do arquivo INI será *atualizador.ini*, o mesmo vai ser gerado no diretório que for executado o atualizador.

Como podemos observar no método *GravarINI*, passamos por parâmetro o nome da tabela, o campo e o valor do campo. Agora vamos utilizar o procedimento para gravar o arquivo de configuração do FTP. No evento *OnClick* do *BitBtn1* adicione o código da **Listagem 3**.

Lendo o arquivo de configuração

Nessa etapa, vamos fazer a leitura do arquivo INI que possui as configurações do atualizador, através da **Listagem 4**.

Componente	Propriedade	Valor
Label1	Name	"lblstatus"
Label2	Name	"lblcontador"
Bitbtn1	Name/Caption	"btnconfigurar"/ "Configurar"
Bitbtn2	Name/Caption	"btnatualizar"/"Atualizar"
Bitbtn3	Name/Caption	"btnfechar"/"Fechar"
Panel1	Color	clWhite
IdFTP1	Name	"ftpupdate"
Form1	Name/Caption	"frmupdate"/"Atualizador"

Tabela 1. Propriedades dos componentes da tela principal

Figura 2. Tela da configuração do cliente FTP

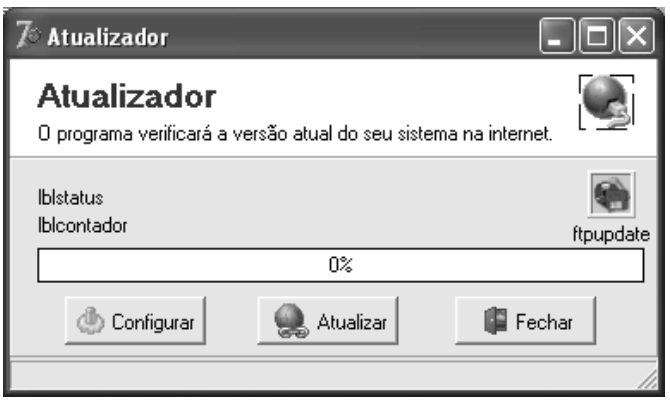


Figura 1. Tela principal do atualizador

Listagem 1. Estrutura do arquivo INI

```
[CONEXAO]
FTP=ftp.dominio.com.br
USUARIO=usuario
SENHA=1234
PASSIVO=N
DIRETORIOCLIENTE=\sistema
ARQUIVO=sistema.exe
```

Listagem 2. Procedimento para criar o arquivo INI

```
procedure Tfrmconfiguracao.GravarINI(
  tabela_ini,campo_ini, valor_ini: string);
var
  ServerIni: TIniFile;
begin
  ServerIni := TIniFile.Create(ExtractFilePath(
    ParamStr(0)) + 'atualizador.ini');
  ServerIni.WriteString(TABELA_INI, CAMPO_INI,
    VALOR_INI);
  ServerIni.UpdateFile;
  ServerIni.Free;
end;
```

Listagem 3. Gravando o arquivo INI

```
procedure Tfrmconfiguracao.BitBtn1Click(
  Sender: TObject);
begin
  gravarini('CONEXAO', 'FTP', edftp.Text);
  gravarini('CONEXAO', 'USUARIO', edusuario.Text);
  gravarini('CONEXAO', 'SENHA', edsenha.Text);
  if Checkbox1.Checked then
    gravarini('CONEXAO', 'PASSIVO', 'S')
  else
    gravarini('CONEXAO', 'PASSIVO', 'N');
  gravarini('CONEXAO', 'DIRETORIOCLIENTE',
    edsalvar.Text);
  gravarini('CONEXAO', 'ARQUIVO', edarquivo.Text);
  gravarini('CONEXAO', 'DIRETORIOSERVIDORFTP',
    eddiretorio.Text);
end;
```

Vamos chamar a função no evento *OnShow* do formulário, trazendo as informações do arquivo INI, carregando os dados para a tela de configuração do FTP (**Listagem 5**).

Para fazer a chamada ao formulário de configuração, abra o *frmupdate* e digite o código da **Listagem 6** no evento *OnClick* do *btnconfiguracao* (não esqueça de declarar a unit *untfrmconfiguracao* no uses do formulário).

Componente de acesso ao servidor FTP

Vimos como manipular o arquivo INI da configuração do FTP, agora conectaremos ao servidor FTP através do *IdFTP*. Utilizaremos *threads* em nosso exemplo, sendo *thread* uma programação concorrente que executa uma ou mais tarefas ao mesmo tempo. Seria bastante ruim deixar o atualizador impossibilitado de efetuar outras tarefas ("congelar").

Listagem 4. Função para leitura do arquivo INI

```
function Tfrmconfiguracao.lerini(tabela_ini,
  campo_ini: string): string;
var
  ServerIni: TIniFile;
begin
  ServerIni := TIniFile.Create(ExtractFilePath(
    ParamStr(0)) + 'atualizador.ini');
  Result := ServerIni.ReadString(tabela_ini,
    campo_ini, '');
  ServerIni.Free;
end;
```

Listagem 5. Utilizando a função para mostrar as informações do arquivo INI na tela

```
procedure Tfrmconfiguracao.FormShow(Sender: TObject);
begin
  edftp.Text := lerini('CONEXAO', 'FTP');
  edusuario.Text := lerini('CONEXAO', 'USUARIO');
  edsenha.Text := lerini('CONEXAO', 'SENHA');
  if lerini('CONEXAO', 'PASSIVO') ='S' then
    Checkbox1.Checked := True
  else
    Checkbox1.Checked := False;
  edsalvar.Text := lerini('CONEXAO',
    'DIRETORIOCLIENTE');
  edarquivo.Text := lerini('CONEXAO', 'ARQUIVO');
  eddiretorio.Text := lerini('CONEXAO',
    'DIRETORIOSERVIDORFTP');
end;
```

Listagem 6. Chamada para o formulário de configuração

```
procedure Tfrmupdate.btnconfigurarClick(
  Sender: TObject);
begin
  frmconfiguracao := Tfrmconfiguracao.Create(Self);
  try
    frmconfiguracao.ShowModal;
  finally
    frmconfiguracao.Release;
    frmconfiguracao := nil;
  end;
end;
```


ClubeDelphi PLUS

Acesse agora o mesmo o portal do assinante ClubeDelphi (www.clubedelphi.net/portal) e assista a uma vídeo-aula de Guinther Pauli, que mostra uma introdução ao uso de Threads no Delphi.

http://www.devmedia.com.br/articles/viewcomp.asp?comp=1733

O primeiro passo para gerar uma *thread* é acessar o *frmupdate* e criar uma classe descendente de *TThread*. Declare uma sessão *protected* e insira o método *Execute*, que é virtual da classe *TThread*, conforme o seguinte código:

```
threadFTP = class(TThread)
protected
  procedure Execute; override;
end;
```

Pressione as teclas CTRL+SHIFT+C para criar o cabeçalho do método, implementando-o conforme a **Listagem 7**.

Na listagem anterior utilizamos o *IdFTP*, que tem como função pegar os dados de configuração do arquivo INI, tais como: host, usuário, senha, modo de conexão e o arquivo que será atualizado. O método *Connect* estabelece conexão com o servidor FTP.

Listagem 7. Método execute da threadFTP

```
procedure threadFTP.Execute;
begin
  with frmupdate do
  begin
    with frmconfiguracao do
    begin
      ftpupdate.Host := lerini('CONEXAO', 'FTP');
      ftpupdate.Username := lerini('CONEXAO',
        'USUARIO');
      ftpupdate.Password := lerini('CONEXAO',
        'SENHA');
      if lerini('CONEXAO', 'PASSIVO') ='S' then
        ftpupdate.Passive := True
      else
        ftpupdate.Passive := False;
      ftpupdate.Connect(true);
      ftpupdate.Changedir(lerini('CONEXAO',
        'DIRETORIOSERVIDORFTP'));
      ftpupdate.List(nil);
      tamanho_arquivo := ftpupdate.Size(lerini(
        'CONEXAO', 'ARQUIVO'));
      if FileExists(lerini('CONEXAO',
        'DIRETORIOCLIENTE')+'\'+lerini('CONEXAO',
        'ARQUIVO')) then
      begin
        if FormatDateTime('dd/mm/yyyy HH:mm',
          FileDateToDateTime(FileAge(lerini('CONEXAO',
          'DIRETORIOCLIENTE')+'\'+lerini('CONEXAO',
          'ARQUIVO')))) <> FormatDateTime(
          'dd/mm/yyyy HH:mm',
          ftpupdate.DirectoryListing.Items[
          0].ModifiedDate)  then
        begin
          if MessageDlg('Existe uma nova versão '+
            'dessa aplicação disponível na internet!'
            +#13+'Deseja atualizar?', mtconfirmation,
            [mbyes, mbno], 0) = mryes then
          begin
            ftpupdate.get(lerini('CONEXAO', 'ARQUIVO'),
              lerini('CONEXAO', 'DIRETORIOCLIENTE')+
              '\'+lerini('CONEXAO', 'ARQUIVO'), True);
            FileSetDate(lerini('CONEXAO',
              'DIRETORIOCLIENTE')+'\'+lerini(
              'CONEXAO', 'ARQUIVO'),
              DateTimeToFileDate(
              ftpupdate.DirectoryListing.Items[
              0].ModifiedDate));
          end;
        end;
      end
      else
      begin
        ftpupdate.get(lerini('CONEXAO', 'ARQUIVO'),
          lerini('CONEXAO', 'DIRETORIOCLIENTE')+'\'+
          lerini('CONEXAO', 'ARQUIVO'), True);
        FileSetDate(lerini('CONEXAO',
          'DIRETORIOCLIENTE')+'\'+lerini('CONEXAO',
          'ARQUIVO'), DateTimeToFileDate(
          ftpupdate.DirectoryListing.Items[
          0].ModifiedDate));
      end;
      ftpupdate.Disconnect;
    end;
  end;
end;
```

ChangeDir é utilizado para posicionar o atualizador no diretório do arquivo localizado no servidor FTP, ou seja, o arquivo para atualizar o sistema pode estar em uma estrutura de pastas dento do FTP, assim utilizamos o método para posicionar na pasta especificada. O método *List* é responsável por listar os arquivos presentes no diretório atual, análogo ao comando dir do velho DOS.

A variável *tamanho_arquivo*, como seu próprio nome sugere, recebe o tamanho do arquivo. Essa informação é fundamental para o funcionamento do *Gauge* (barra de progresso) no processo de download, que será apresentado a seguir.

O *Get* é responsável pelo download do arquivo, onde o arquivo a ser copiado e o destino do mesmo são parâmetros desse método. Antes de inicializar o download verifica-se a versão do sistema cliente com a disponível no servidor, sendo diferentes o sistema mostrará uma mensagem, com o objetivo de verificar se o usuário deseja atualizar o sistema (algo semelhante ao que acontece por exemplo em softwares como o MSN).

No evento *OnClick* do botão *Atualizar* digite o seguinte código, que instancia a *thread* e passa por parâmetro *False*, para que a mesma seja executada no momento em que for criada.

```
threadFTP.Create(False);
```

Status e movimentação do Gauge

A aplicação está praticamente pronta, entretanto não existem informações sobre o processo de download. Dessa forma, utilizaremos o *IdFTP* para sanar esse problema. O componente possui o evento *OnStatus* responsável por mostrar a situação do processo. Assim, mostraremos na *StatusBar* o status da conexão ao usuário, adicionando o código da **Listagem 8** no evento *OnStatus* do *ftpupdate*.

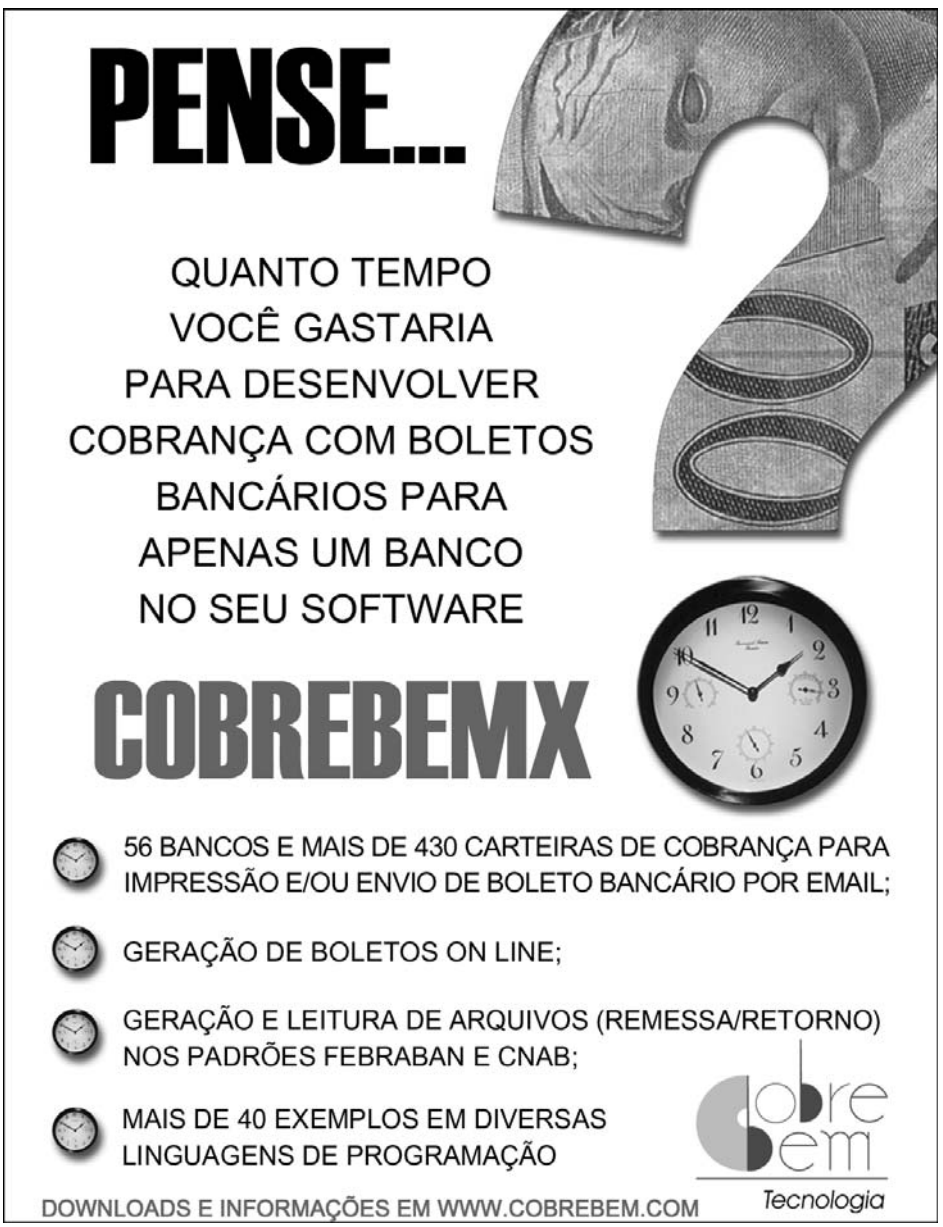
Para deixar aplicação mais elegante, indicaremos o progresso do download no *Gauge*, o mesmo vai mostrar o percentual de transferência do arquivo. Para isso, utilizamos os eventos *OnWork* e *OnWorkBegin* do *ftpupdate*, e precisamos declarar algumas variáveis na seção private:

```
STime: TDateTime;
tempo_medio: double;
bytes_transf: longword;
tamanho_arquivo: longword;
```

O evento *OnWork* do *IdFTP* recebe o número de bytes que está sendo transferido no momento do download ou upload do servidor FTP através do

Listagem 8. Evento para mostrar o status do FTP

```
procedure Tfrmproutupdate.ftpupdateStatus(
  ASender: TObject; const AStatus: TIdStatus;
  const AStatusText: string);
begin
  case aStatus of
    hsResolving: Statusbar1.Panels[0].Text :=
      'Resolvendo..';
    hsConnecting: Statusbar1.Panels[0].Text :=
      'Conectando..';
    hsConnected: Statusbar1.Panels[0].Text :=
      'Conectado ao Servidor FTP! Aguarde..';
    hsDisconnecting: statusbar1.Panels[0].Text :=
      'Desconectando..';
    hsDisconnected: statusbar1.Panels[0].Text :=
      'Desconectado!';
    ftpTransfer: statusbar1.Panels[0].Text :=
      'Transferindo..';
    ftpReady: statusbar1.Panels[0].Text := 'Lendo...';
    ftpAborted: statusbar1.Panels[0].Text :=
      'Abortado!';
  else
    StatusBar1.Panels[0].Text := AStatusText;
  end;
end;
```



parâmetro *AWorkcount*. Agora, veremos o motivo do uso da variável *tamanho_arquivo*, pois alguns servidores de FTP não informam o tamanho do arquivo no início da transferência. Assim, usamos o *tamanho_arquivo* para suprir essa necessidade, o mesmo será utilizado para efetuar o cálculo do percentual de dados transferidos, a ser apresentado no *Gauge*.

Confira na **Listagem 9** o código do evento *OnWork*, cuja função é calcular a taxa média, o tamanho do arquivo e o percentual de transferência que será visualizado no *Gauge*.

Listagem 9. Código que movimenta o Gauge usando o evento OnWork

```
procedure Tfrmupdate.ftpupdateWork(Sender: TObject;
  AWorkMode: TWorkMode; const AWorkCount: Integer);

var
  contador, kbTotal, kbTransmitidos,
    kbFaltantes: Integer;
  Status_trans: string;
  TotalTempo: TDateTime;
  H, M, Sec, MS: Word;
  DLTime, media: Double;

begin
  kbTotal := tamanho_arquivo div 1024;
  TotalTempo :=  Now - STime;
  DecodeTime(TotalTempo, H, M, Sec, MS);
  Sec := Sec + M * 60 + H * 3600;
  DLTime := Sec + MS / 1000;
  KbTransmitidos := AWorkCount div 1024;
  kbFaltantes := kbTotal - kbTransmitidos;
  lblcontador.Caption := 'Transferidos: '+
    FormatFloat('##,###,##0', kbTransmitidos ) +
    ' Kb de ' + FormatFloat('##,###,##0', kbTotal) +
    ' Kb';
  media := (100/tamanho_arquivo) * AWorkCount;

  if DLTime > 0 then
  begin
    tempo_medio := (AWorkCount / 1024) / DLTime;
    Status_trans := Format('%2d:%2d:%2d',
      [Sec div 3600, (Sec div 60) mod 60,
      Sec mod 60]);
    Status_trans := 'Tempo de download ' +
      Status_trans;
  end;

  Status_trans := 'Taxa de Transferencia: '+
    FormatFloat('0.00 KB/s', tempo_medio) + '; ' +
    Status_trans;
  lblstatus.Caption := Status_trans;
  Application.ProcessMessages;
  Contador := Trunc(media);
  Gauge1.Progress := (contador);

end;
```

Na listagem anterior foi utilizado cálculo da cinemática. No evento *OnWorkBegin* do *ftpupdate*, para calcular a taxa e o tempo de transferência, devemos inicializar as variáveis conforme o seguinte código:

```
Stime := Now;
tempo_medio := 0;
```

Para concluir o 100% do *Gauge* e informar no status que o download foi concluído, no evento *OnDisconnected* do *ftpupdate*, digite o seguinte código:

```
Gauge1.AddProgress(100);
StatusBar1.Panels[0].Text :=
  'Download concluído!! Desconectado do servidor FTP.';
```

Veja na **Figura 3** a aplicação em execução.

Conclusão

A internet é um mundo fantástico, usamos uma pequena fatia para desenvolver a nossa aplicação, através do protocolo FTP, que foi a peça fundamental para automatizar o processo de atualização do sistema no cliente.

Uma aplicação simples, mas de grande valia para o cliente e o desenvolvedor. Ambas as parte estarão na mesma sintonia, aplicando as atualizações que possam ocorrer e aproveitando todos os recursos que foram atualizados. ■

Figura 3. Aplicação atualiza sistema automaticamente, via FTP

Como melhorar o desempenho do BDS

FELLIPE HENRIQUE
(fellipeh@gmail.com)
é programador e desenvolvedor em Delphi há mais de oito anos e graduando em Sistemas de Informação pela Faculdade de Minas – FAMINAS. Trabalha principalmente com tecnologia multi-camadas e cliente/servidor.

Nessa nova versão do *Borland Developer Studio* (daqui para frente referenciaremos apenas como BDS), vimos que a Borland trouxe diversas opções incluídas de forma a aperfeiçoar nosso trabalho, dentre elas podemos destacar o *Together*, *StarTeam*, *CaliberRM* entre outros.

Porém, vários desenvolvedores não utilizam todas essas novidades, e com isso elas acabam prejudicando nosso trabalho, fazendo com que o BDS fique lento. Neste artigo mostrarei como você poderá otimizar o BDS, removendo essas opções, deixando-o básico e por conseqüência mais rápido.

Antes de iniciar todo o procedimento, você deverá efetuar um backup geral do registro do Windows, pois serão necessárias algumas modificações. Feito isso, mãos à obra.

Preparando o ambiente (criando o seu perfil)

Para iniciarmos, você deverá criar um atalho para a execução do BDS, para isso, vá em *Iniciar>Borland Developer Studio* e em seguida clique com o botão direito no item *Delphi for Microsoft Win32*. Selecione *Enviar para* e clique em *Área de trabalho (criar atalho)*.

Após todo o processo descrito, clique com o botão direito no ícone na *Área de Trabalho* e selecione *Propriedades*. Na tela de propriedades, selecione a aba *Atalho* e modifique o campo *Destino*, inserindo ao final da *string*: "-rNOME" (**Figura 1**).

> **Dica:** Troque o *-rNOME*, pelo nome que você deseja criar o perfil, exemplo: "-rFellipe".

Após a modificação, finalize clicando em OK. Depois, execute o BDS clicando duas vezes no ícone recém modificado, quando o BDS abrir, feche-o. Com isso, você acabou de criar um perfil para o BDS e é nele que iniciaremos nossa modificação.

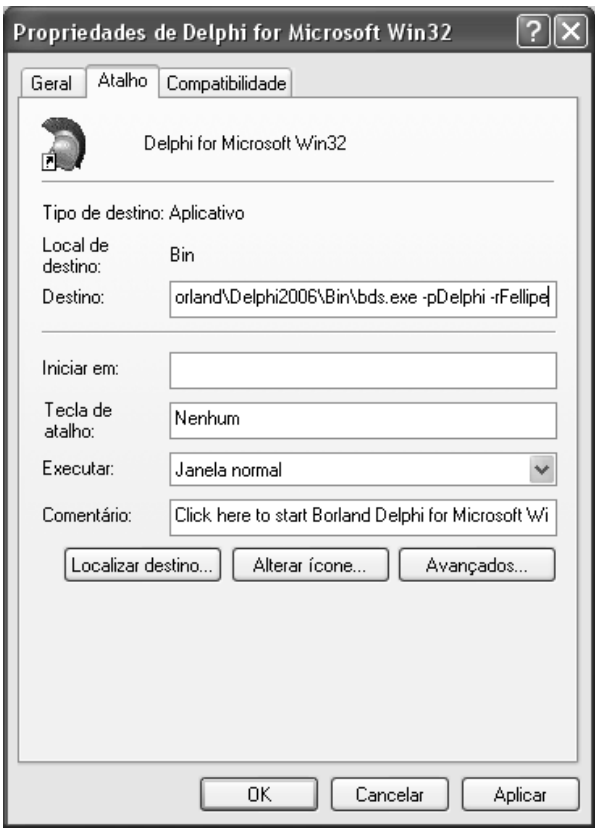


Figura 1. Modificando a propriedades do atalho.

Modificando os pacotes

Com o BDS fechado, modificaremos o registro do Windows. Para isso, selecione *Executar* no menu *Iniciar* e digite "regedit". Com o registro aberto, acesse o seguinte caminho: HKEY_CURRENT_USER/Software/Borland.

É nessa pasta que encontram-se todas as configurações do BDS. Dentro dela estará a subpasta com o nome do perfil que você escolheu (**Figura 2**).

Acesse essa pasta e depois em *4.0* (referente ao BDS). Dentro de *4.0* temos todas as configurações referente ao perfil. Trataremos somente das seguintes pastas: *Known IDE Packages* e *Known Packages*. Para maiores explicações sobre as pastas, veja a **Tabela 1**.

Pasta	Descrição
Experts	Todos os experts registrados.
Known Assemblies	Todos os Assemblies.
Known IDE Packages	Todos os pacotes que fazem parte da IDE do BDS.
Known Packages	Todos os pacotes de componentes registrados.
Library	Todos as pastas cadastradas na Library do BDS.

Tabela 1. Explicações sobre as pastas

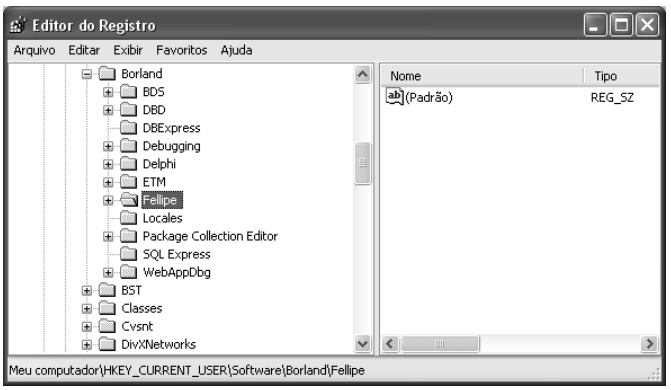


Figura 2. Pasta criada com o perfil usado no atalho

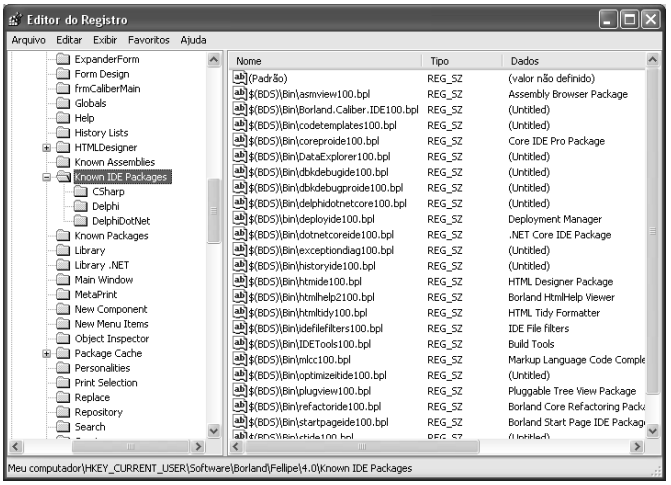


Figura 3. Chaves de registros dos pacotes do Delphi

Nossa preocupação no momento é com *Known IDE Packages*. Como visto na **Tabela 1** é referente aos pacotes que serão iniciados junto com o BDS, clicando nele você perceberá que no lado direito, temos várias chaves de registro (**Figura 3**).

Cada chave de registro é um pacote que será iniciado junto com o BDS, removendo algum, esse não será mais iniciado. A **Tabela 2** mostra as descrições de algumas chaves e cabe ao desenvolvedor remover a que mais lhe convém.

Chave	Explicação
$(BDS)\Bin\Borland.Caliber.IDE100.bpl	Integração do CaliberRM.
$(BDS)\Bin\codetemplates100.bpl	Integração dos Templates.
$(BDS)\Bin\DataExplorer100.bpl	Integração do Data Explorer.
$(BDS)\Bin\historyide100.bpl	Histórico dos Arquivos.
$(BDS)\Bin\optimizeitide100.bpl	Integração do OptimizeIt.
$(BDS)\Bin\refactoride100.bpl	Integração do novo Refactor.
$(BDS)\Bin\startpageide100.bpl	Página inicial (Welcome Page).
$(BDS)\Bin\stide100.bpl	Integração do StarTeam.
$(BDS)\Bin\todoide100.bpl	To-Do padrão do Delphi.

Tabela 2. Explicações sobre os pacotes iniciados junto com o BDS

Como estamos lidando com a personalização do Delphi for

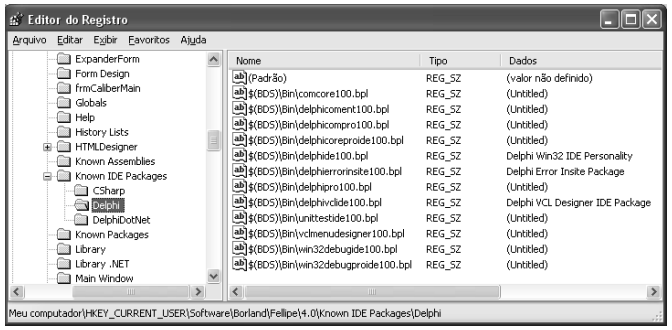


Figura 4. Pacotes somente para Win32

Win32, devemos ir à subpasta *Delphi*, dentro de *Known IDE Packages*. Sendo encontrados alguns pacotes especiais somente para Delphi, ou seja, serão iniciados somente pelo Win32 (**Figura 4**).

A **Tabela 3** tem as explicações de alguns deles, como visto anteriormente a remoção de alguma chave, resulta na não inicialização do pacote.

Chave	Explicação
$(BDS)\Bin\delphierrorinsite100.bpl	Integração do novo Error Insite.
$(BDS)\Bin\TGIDE100.bpl	Integração do Together. Ao remover esse pacote, o Model View também não será executado.

Tabela 3. Pacotes iniciados para Win32

Prontinho! Feito isso, execute o BDS através do ícone criado e perceberá a grande diferença. Porém, caso você tenha algum componente instalado, você notará que esses não foram carregados. Isso ocorre porque ao criar um novo perfil, o BDS cria um perfil básico, ou seja, somente com os componentes padrões do próprio BDS.

Uma dúvida que poderá surgir: terei que reinstalar todos os meus componentes novamente? Bem, essa é uma opção, mas se você for como eu e tiver mais de 30 bibliotecas ficará trabalhoso reinstalar todos novamente. Para isso temos uma solução mais prática.

Importando os componentes já instalados.

Como você deve ter percebido no início do artigo, temos a pasta chamada *Known Packages*, e é nela que estão todos os componentes instalados. Para que não tenhamos nenhum componente instalado em duplicidade, acesse *Known Packages* do perfil recém criado e excluiremos todas as chaves, **exceto** a chave *Padrão*.

Após isso, devemos procurar a pasta chamada *BDS*, onde estará localizada a pasta *4.0*. Essa é a pasta padrão do Delphi, por isso não modifique nada. Localize a pasta chamada *Known Packages* e você poderá perceber que todos os seus componentes estarão lá. Para exportar as chaves, clique com o botão direito em *Known Packages* e selecione *Exportar*, digite o nome do arquivo e pronto.

Após esse processo, acesse o arquivo recém criado, clique com o botão direito e selecione *Editar*. Com o *Bloco de Notas* aberto, podemos perceber que a terceira linha é o caminho no registro, que estará assim: [HKEY_CURRENT_USER\Software\Borland\BDS\4.0\Known Packages].

Devemos modificar para: [HKEY_CURRENT_USER\Software\Borland\NOME_PERFIL\4.0\Known Packages], onde NOME_PERFIL é o nome que você escolheu para o seu perfil. Após isso salve o arquivo e execute-o clicando duas vezes em cima dele.

Com isso serão adicionados todos os pacotes no registro do Windows e por conseqüência no BDS. Execute o BDS e você verá que todos os seus componentes estarão lá.

> **Dica:** Tudo que você fizer no seu perfil, não influenciará em nada o BDS padrão, caso você queira executar o BDS de forma normal, basta executar pelo atalho original da instalação.

Otimizando para .NET

Para otimizar para Delphi for .NET, proceda da mesma forma como descrito anteriormente, o que mudará é a subpasta de *Known IDE Packages*, que devemos selecionar *DelphiDotNet* e proceder com a mesma técnica descrita anteriormente.

Conclusão

Vimos neste artigo como é fácil modificar nosso BDS para que fique mais rápido e de forma que agrade. Vale lembrar que como modificamos o registro do Windows, é sempre bom ter um backup atualizado, e como manda o figurino: é sempre bom ter cautela ao remover os pacotes originais, por isso, vá removendo aos poucos e executando o BDS para que você sinta a diferença. ■

Complete sua coleção!

Feita para Desenvolvedores de Software

SQL magazine

Escolha 03 edições e pague **R$ 6,90** em cada revista.

20

- Mapeamento Objeto Relacional. Aprenda a construir um mapeador O/R em .NET.

- Warehouse Builder. Construa uma data warehouse com a ferramenta ETL da Oracle-Partel

- UML. Aprenda passo a passo como construir um diagrama de sequências.

- Otimização. Influenciando o otimizador de consultas.

21

- Normalização. Aprenda a organizar melhor o esquema de seu banco de dados

- PostreSQL. Saiba o que são rules e como utilizá-las.

- SQL Server 2000. Entenda os modelos de recovery.

- InterBase 7.5. Quando e porque migrar.

- Particionamento do Oracle. Perigos e armadilhas Parte II.

22

- Backup. Saiba como proteger seus dados no Oracle e no SQL Server.

- UML. Conheça algumas opções de ferramentas livres - Parte II.

- Homologação de dados. Saiba como avaliar a qualidade de seu modelo de dados.

- Especial. Acessando servidores Oracle a partir do SQL Server 2000.

23

- Proteja-se. Saiba como funciona o SQL Injection e aprenda a lidar com ele no SQL Server.

- Administrando o Oracle. Conheça o Oracle Enterprise Manager.

- Backup. Saiba como efetuar o restore de Bds no SQL Server.

- Modelagem. Utilizando padrões na elaboração do projeto de banco de dados.

24

- Java + BD + Web. Trabalhe de forma integrada com: MySQL, Tomcat e Java com NetBeans.

- Oracle 10g. Aumente a disponibilidade de seu SGBD com o automatic Storage Management.

- PostgreSQL. Veja na prática como efetuar replicação entre servidores PostgreSQL.

25

- Clustering no SQL Server. Saiba para que serve e como funciona essa tecnologia de redundância.

- IBExpert. Configure e administre base de dados no Firebird e interbase.

- Caché. trabalhe com objetos diretamente no banco de dados.

- Oracle Discoverer. A prenda passo a passo a trabalhar com a ferramenta de análise de dados da Oracle.

26

- Mineração de dados. Entenda como ocorre a descoberta de informações em base de dados.

- IBExpert. Administrando o InterBase/Firebird. Conheça algumas características avançadas.

- UML. Ferramenta de modelagem free com engenharia reversa.

- MySQL. Transforme-se em um especialista na otimização de consultas - Parte II.

27

- MySQL. Veja como rodar um banco de dados MySQL em várias máquinas ao mesmo tempo!

- ER/Studio. Conheça esta excelente ferramenta para modelagem de dados.

- SQL. Descobrindo associações entre pares de linhas.

- SQL Server. Veja como enviar e-mails anexos no SQL Server sem o uso do SQL Mail.

28

- Dúvidas frequentes em bancos de dados.

- Desafio SQL. Construindo um modelo para um jogo de futebol on-line.

- Index Tuning Wizard. Como produzir tunings automáticos.

- Erwin conheça esta ferramenta de modelagem E/R.

29

- Oracle FLASHBACK. Recupere informações para um determinado ponto do tempo.

- SQL Server. Descubra se você esta trabalhando corretamente com o SQL Server.

- InterBase / Firebird. Trabalhando com stored procedures e triggers.

www.devmedia.com.br/msdn/anteriores

ou pela nossa central de atendimento: (21)2220-5375

Saiba como trabalhar corretamente com o NULL no Firebird

PAUL VINKENOOG

Repetidas vezes, perguntas de suporte surgem nas listas de e-mail do Firebird relatando que "coisas estranhas" acontecem com NULLs. O conceito parece difícil de captar, em parte, possivelmente, por causa do nome, que sugere aparentemente um "nada", não causará nenhum problema, caso o acrescentemos a um número ou o adicionarmos ao final de uma *string*.

Na verdade, a execução de tais operações fará com que o resultado final seja um NULL. Este artigo mostrará o comportamento do NULL no Firebird, aponta as armadilhas mais comuns e mostra como lidar com segurança, com expressões que contêm NULL ou podem resultar em um NULL.

Observação: Algumas sentenças e exemplos neste artigo, foram extraídos do *Firebird Quick Start Guide*, publicado primeiramente por IBPhoenix e que atualmente faz parte do projeto Firebird.

O que é NULL?

No SQL, NULL não é um valor, é, porém, o *estado* que indica que o valor de um item é desconhecido ou inexistente. Não é o valor zero, o espaço ou "uma *string* vazia", e não se comporta como nenhum desses valores.

Poucas coisas no SQL levam a maior confusão do que o NULL, no entanto, seu funcionamento não deverá ser difícil de entender enquanto adotamos essa simples definição: NULL significa *desconhecido*. Permita-me repetir:

NULL significa DESCONHECIDO

Tendo essa frase em mente no restante do artigo, a maior parte dos resultados aparentemente ilógicos obtidos com o uso do NULL, irão se explicar praticamente por si mesmo.

NULL em expressões

Assim como muitos, já descobrimos que o NULL é contagioso: seja utilizado em um numérico, em *string* ou em expressões data/hora, o resultado sempre será NULL. Utilizado em uma expressão booleana, o resultado depende do tipo da operação e dos outros valores implicados.

Note que em versões Firebird anteriores à 2.0 é geralmente ilegal utilizar constantes NULL diretamente em operações ou comparações. Onde quer que o NULL apareça nas expressões a seguir, leia-se como "um campo, uma variável ou outra expressão que resulta em NULL".

Expressões que retornam NULL

As expressões da **Listagem 1** *sempre* retornam NULL:

Listagem 1. Expressões que sempre retornam NULL

```
1 + 2 + 3 + NULL
'Home ' || 'sweet ' || NULL
MyField = NULL
MyField <> NULL
NULL = NULL
not (NULL)
```

Se houver alguma dificuldade para entender o porquê, basta lembrar que NULL significa "desconhecido". Vejamos também a **Tabela 1**, onde foram fornecidas explicações caso a caso.

Nessa tabela não escrevemos NULL nas expressões (como já foi dito, isso é normalmente ilegal), em lugar disso, utilizamos duas entidades A e B que são ambas NULL. A e B podem ser campos, variáveis ou subexpressões completas por si próprias, desde que sejam NULL, todas irão se comportar da mesma maneira nas expressões.

NULL em expressões booleanas

Já vimos que *not (NULL)* resulta em NULL. Para os operadores *and* e *or*, as coisas são um bocado mais complicadas, conforme podemos ver na **Listagem 2**.

Listagem 2. Expressões utilizando and e or

```
NULL or false = NULL
NULL or true = true
NULL or NULL = NULL
NULL and false = false
NULL and true = NULL
NULL and NULL = NULL
```

O SQL do FireBird não possui o tipo de dados booleano, nem tampouco as constantes existentes são verdadeiras ou falsas. A coluna da esquerda da **Tabela 2**, *true* e *false* representam subexpressões que retornam *true/false*.

Todos esses resultados estão em conformidade com a lógica booleana. O fato de que para calcular "X or true" e "X and false", simplesmente *não temos* que conhecer o valor de X, é também a base das funcionalidades que conhecemos em várias linguagens de programação: o "curto-circuito" da avaliação booleana.

ClubeDelphi PLUS

Acesse agora mesmo o Portal do Assinante ClubeDelphi e assista a uma vídeo-aula de Paulo Quicoli que mostra como instalar o Firebird 2.0 juntamente com a versão antigo (Firebird 1.5).

www.devmedia.com.br/articles/viewcomp.asp?comp=3840

Se A e B são NULL	Será	Por que
1 + 2 + 3 + A	NULL	Se A for desconhecido, então 6 + A também será desconhecido
Home || sweet || A	NULL	Se A for desconhecido, Home sweet || A será desconhecido
MyField = A	NULL	Se A for desconhecido, não poderemos dizer que MyField terá o mesmo valor...
MyField <> A	NULL	... porém, também não poderemos dizer que MyField tem um valor diferente!
A = B	NULL	Se A e B são desconhecidos, será impossível saber se são iguais.
not (A)	NULL	Se A é desconhecido, seu recíproco também o é.

Tabela 1. Operações sobre as entidades nulas A e B

Se A é NULL	Será	Por que
A or (false)	NULL	"A or false" sempre tem o mesmo valor de A, que é desconhecido.
A or (true)	true	"A or true" é sempre true, o valor de A não tem importância.
A or A	NULL	"A or A" sempre é igual a A, que é NULL.
A and (false)	false	"A and false" é sempre false, o valor de A não tem importância.
A and (true)	NULL	"A and true" sempre tem o mesmo valor de A, que é desconhecido.
A and A	NULL	"A and A" sempre é igual a A, que é NULL.

Tabela 2. Operações booleanas em entidade nula A

NULL em funções agregadas

Em funções agregadas tais como COUNT, SUM, AVG, MAX e MIN, o NULL é tratado de formas diferentes, para calcular o resultado, somente os campos não NULL são levados em consideração. Isso é, se tivermos os dados de uma tabela chamada *MyTable* com os valores, constantes na **Tabela 3**.

ID	Name	Amount
1	John	37
2	Jack	<NULL>
2	Joe	5
4	Josh	12
5	Jay	<NULL>

Tabela 3. Dados de uma tabela para exemplo de funções agregadas

O código:

```
select sum(Amount)from MyTable
```

Retornará 54, equivalente a 37 + 5 + 12. Caso os cinco campos fossem somados, o resultado teria sido NULL. Para a função AVG, os campos não NULL são somados e a soma dividida pelo número de campos não NULL, de acordo com os dados da tabela anterior, seria 18.

Existe uma exceção a essa regra: COUNT (*) retorna a contagem de todas as linhas, inclusive linhas cujos campos são todos NULL. No entanto, a função COUNT (*FieldName*), comporta-se como outras funções agregadas em que somente serão contadas as linhas em que o campo especificado não forem NULL.

Outro fato valioso que devemos ter conhecimento, é que o COUNT (*) e o COUNT (*FieldName*) nunca retornam NULL. Se não existir nenhuma linha no conjunto, ambas as funções retornarão "0". A função COUNT (*FieldName*), também retorna "0" caso todos os campos *FieldName* no conjunto sejam NULL.

Outras funções agregadas retornam NULL em tais casos. Devemos estar cientes de que até a função SUM retorna NULL, caso utilizada em um conjunto vazio, o que representa o contrário da lógica normal.

Manipulação do NULL em UDFs

As *UDFs* (*User Defined Funtions* ou *Funções Definidas pelo Usuário*), são funções que não estão embutidas no engine do banco de dados, porém são definidas em módulos separados. O Firebird é distribuído com duas bibliotecas UDF: *ib_udf* (herdada do InterBase) e *fbudf*.

Podemos acrescentar bibliotecas adicionais, por exemplo, comprando bibliotecas de terceiros ou escrevendo as nossas próprias. As UDFs não podem ser utilizadas independentemente, têm que ser antes "declaradas" no banco de dados. Isso é válido também para UDFs distribuídas com o próprio Firebird .

Conversões NULL e NOT NULL indesejadas

Ensinar como declarar, escrever e utilizar UDFs, está fora do escopo deste artigo. No entanto, devemos alertar que as UDFs podem ocasionalmente, realizar conversões NULL inesperadas. Isso faz com que às vezes, entradas NULL sejam convertidas para um valor normal e que em outras situações, conduzam à anulação de uma entrada válida, tal como uma *string* vazia ('').

A principal causa para esse problema é que com "o velho estilo" de chamadas a UDFs, não é possível passar NULL como a entrada para a função. Quando chamamos uma UDF tal como a LTRIM (*left trim*) com um argumento NULL, o argumento é passado para a função como uma *string* vazia.

Internamente, não existe *nenhuma maneira* de saber se esse argumento representa uma *string* vazia ou um NULL. Portanto, o que o executor da função faz? Tem que fazer uma escolha: ou tomar o argumento pelo seu valor nominal ou assumir que era originalmente um NULL e tratá-lo de acordo.

Dependendo do tipo do resultado, retornar NULL pode ser possível mesmo que receber um NULL. Portanto, os seguintes eventos inesperados podem acontecer:

- Chamamos uma UDF com um argumento NULL, que é passado como um valor, por exemplo, "0" ou " (*string* vazia). Dentro da função, esse argumento não é alterado para NULL, um resultado não NULL será retornado;
- Chamamos uma UDF com um argumento válido, tal como "0" ou ", que é passado sem alterações (obviamente). No entanto, o código da função supõe que esse valor, na verdade representa um NULL, o trata como

um "buraco negro" e retorna NULL para o chamador.

Ambas as conversões são normalmente indesejadas, porém, a segunda provavelmente é menos ainda do que a primeira (melhor validar algo NULL do que destruir algo válido). Retornando ao exemplo do LTRIM, até o Firebird 1.0.3, essa função retornava NULL caso fosse alimentada com uma *string* vazia.

A partir da versão 1.5, nunca retorna NULL. Nas versões mais recentes, seqüências NULL são "aparadas" para *strings* vazias, o que está errado, porém, é considerado o menor dos males: na antiga situação, seqüências válidas (vazias) eram impiedosamente resolvidas como NULL.

Preparado para conversões indesejadas

As conversões indesejadas descritas anteriormente, normalmente só acontecem com UDFs legadas, porém, essas existem ainda em grande quantidade (notavelmente na *ib_udf*). Também, nada impedirá que um programador descuidado, faça o mesmo em uma nova função.

Portanto, devemos tomar os seguintes cuidados, caso utilizemos uma UDF que não conhecemos detalhadamente o seu comportamento em relação ao NULL:

1. Investigar sua declaração para descobrir como os valores são passados e retornados. Caso seja "por descritor", então deverá ser segura (embora não custe garantir). Em todos os outros casos, devemos percorrer o resto dos passos;

2. Caso tenhamos o fonte da UDF e conhecermos a linguagem C/C++, devemos inspecionar o código de função;

3. Testar a função tanto com entradas NULL quanto com entradas do tipo "0" (para argumentos numéricos) e/ou '' (para argumentos *string*).

4. Caso a função execute uma conversão NULL <> not NULL indesejada, deveremos contorná-la via código antes de chamar a UDF (ver tópico *Testando se algo é NULL*, neste artigo).

As declarações das bibliotecas UDF nativas podem ser encontradas no subdiretório *bin/examples* do Firebird (1.0) ou *bin/UDF* (1.5 e superior), através dos arquivos com extensão SQL.

NULL em If

Se a expressão de teste de uma declaração (*if*) produzir um NULL, então o bloco *if* da declaração será pulado e a cláusula *else* (se presente) será executada. No entanto, devemos ter cuidado!

A expressão, nesse caso, pode *comportar-se* como *false*, porém, pode não ter *o valor false*, portanto será ainda NULL e coisas esquisitas podem acontecer se esquecermos isso. Os seguintes exemplos exploram um pouco do funcionamento "dia-

bólico" do NULL em declarações *if*:

```
if (a = b) then
  MyVariable = 'Equal';
else
  MyVariable = 'Not equal';
```

Se *a* e *b* forem ambos NULL, *MyVariable* ainda será *Not Equal* depois de executar esse código. A razão para isso, é que "a = b" produz NULL, caso pelo menos um dos operandos for NULL.

Sendo o resultado do teste da expressão NULL, o bloco *if* será pulado e o bloco *else* será executado. Vejamos outro exemplo:

```
if (a <> b) then
  MyVariable = 'Not equal';
else
  MyVariable = 'Equal';
```

Aqui, *MyVariable* será *Equal* caso *a* for NULL e *b* não o for e vice-versa. A explicação é análoga a do exemplo anterior. Outro exemplo:

```
if (not (a <> b)) then
  MyVariable = 'Equal';
else
  MyVariable = 'Not equal';
```

O código anterior, aparentemente deveria conduzir aos mesmos resultados que o exemplo anterior, não é? No final das contas, invertemos a expressão de teste e trocamos as cláusulas *if* e *else*.

De fato, enquanto nenhuma variável for NULL, ambos os fragmentos de código são equivalentes. No entanto, assim que *a* ou *b* se tornarem NULL, assim acontecerá com a expressão de teste inteira, a cláusula *else* será executada e o resultado será *Not Equal*.

> **Observação:** Naturalmente estamos cientes de que o terceiro exemplo é totalmente equivalente ao primeiro. Foi incluído simplesmente para realçar mais uma vez, que not (NULL) é equivalente a NULL.

Desse modo, em situações onde a expressão de teste produz NULL, o *not()* não irá invertê-lo.

Testando se algo é NULL

Considerando a destruição de que o NULL é capaz, normalmente desejaremos saber se algo é NULL *antes* de utilizá-lo em uma expressão. Para alguns, o teste óbvio pareceria ser:

```
if (A = NULL) then
```

E de fato existem banco de dados que suportam essa sintaxe para determinar a nulidade. No entanto, o padrão SQL não o permite e o Firebird tampouco. Em versões anteriores à 2.0, mesmo essa sintaxe é totalmente ilegal.

A partir da versão 2.0 em diante, essa sintaxe é permitida, porém, a comparação sempre retorna NULL, independentemente do estado de A e do seu valor. É, por tanto, inútil utilizá-la como um teste de nulidade, o que necessitamos é de um resultado verdadeiro ou falso inconfundível. A maneira correta de testar o NULL é:

```
is null/ ...is not null
```

Esses testes sempre retornam verdadeiro ou falso sem complicações. Veja os exemplos da **Listagem 3**.

Listagem 3. Exemplos de comparação de NULL

```
if (MyField is null) then
...
select * from Pupils
where PhoneNumber is not null
/* retorna os dados de Pupils, onde PhoneNumber não for null */
select * from Pupils
where not (PhoneNumber is null)
/* faz o mesmo que o exemplo anterior */
update Numbers set Total = A + B + C
where A + B + C is not null
```

Poderíamos dizer que ao passo que "=" (quando utilizado como um operador de igualdade) somente pode comparar valores, ao passo que *is* testa estados. Configurando um campo ou variável como NULL.

Os campos e as variáveis podem ser configurados como NULL utilizando a mesma sintaxe do que os valores normais, conforme vemos na **Listagem 4**.

Listagem 4. Configurando campos como NULL

```
insert into MyTable values (1, 'teststring', NULL,
  '8-May-2004')
update MyTable set MyField = null
where YourField = -1
if (Number = 0) then
  MyVariable = null;
```

Mas não foi dito que *MyField = NULL* é ilegal?! Isso está correto, para o *operador de comparação* "=" (pelo menos em versões Firebird pré 2.0). No entanto, aqui estamos falando a respeito de "=" como um *operador de atribuição*.

Infelizmente, ambos os operadores utilizam o mesmo símbolo em SQL. Em atribuições, sendo feitas com "=" ou com lista de inserção, poderíamos tratar o NULL como qualquer outro valor, nenhuma sintaxe especial seria necessária (ou de fato possível).

Trabalhando com o NULL

Essa seção contém algumas dicas e exemplos práticos que podem ser utilizados no trabalho diário com o NULL. Normalmente, não precisaremos adotar medidas especiais para campos ou variáveis que possam ser NULL. Por exemplo, se fizermos o seguinte código:

```
select * from Customers
where Town = 'Ralston'
```

Provavelmente não desejaremos recuperar os clientes cuja cidade não foi especificada. De mesmo modo que:

```
if (Age >= 18) then
  CanVote = 'Yes'
```

Não incluirá pessoas com idade desconhecida, que é também defensível. No entanto:

```
if (Age >= 18) then
  CanVote = 'Yes';
else
  CanVote = 'No';
```

Parece menos correto, caso não soubermos a idade de uma pessoa, não deveremos negar-lhe explicitamente o direito de votar. Pior, seria o seguinte código:

```
if (Age < 18) then
  CanVote = 'No';
else
  CanVote = 'Yes';
```

Não terá o mesmo efeito que o anterior. Se algumas das idades NULL forem em verdade menores do que 18, estaremos agora permitindo o voto de menores! A abordagem correta, seria testar o NULL explicitamente, conforme o código da **Listagem 5**.

Listagem 5. Testando explicitamente um NULL

```
if (Age is null) then
  CanVote = 'Unsure';
else
begin
  if (Age >= 18) then
    CanVote = 'Yes';
  else
    CanVote = 'No';
end
```

Descobrindo se os campos têm o mesmo conteúdo

Às vezes desejamos determinar se dois campos ou variáveis tem o mesmo conteúdo e desejamos considerá-los iguais caso ambos sejam NULL. O teste correto para isso é:

```
if (A = B or A is null and B is null) then
```

Ou, se quisermos tornar explícita a precedência das operações:

```
if ((A = B) or (A is null and B is null)) then
```

Uma palavra de alerta, no entanto, é se A e B forem NULL, a expressão de teste torna-se NULL e não *false*. Isso está correto

em uma declaração *if* e poderíamos inclusive acrescentar uma cláusula *else*, que será executada se A e B não forem iguais (inclusive quando um for NULL e o outro não), com o seguinte código:

```
if (A = B or A is null and B is null) then
  /* A for igual a B */
else
  /* A e B forem diferentes */
```

Porém, não deveremos implementar a brilhante idéia de inverter a expressão e utilizá-la como um teste de desigualdade (como fizemos uma vez), como o seguinte exemplo:

```
/* Não fazer isto! */
if (not(A = B or A is null and B is null)) then
  /* A e B forem diferentes */
```

O código anterior funcionará corretamente caso A e B forem ambos NULL ou ambos não NULL. No entanto, não conseguirá executar a cláusula *else*, se apenas um for NULL. Se quisermos que algo seja feito somente quando A e B forem diferentes, deveremos utilizar uma das expressões corretas mostradas acima e escrever uma declaração "dummy" (sem sentido) na cláusula *then* ou utilizar essa expressão de teste mais longa:

```
/* este é um teste de desigualdade correto: */
if (A <> B or A is null and B is not null or
  A is not null and B is null) then
```

Descobrindo se um campo foi modificado

Em *Triggers*, é geralmente útil saber se certo campo foi modificado (inclusive passado de NULL para NOT NULL ou vice-versa) ou permanece o mesmo. Isso é apenas um caso especial do teste da (des) igualdade de dois campos. Apenas use *New.Fieldname* e *Old.Fieldname* para A e B:

```
if (New.Job = Old.Job or New.Job is null and
  Old.Job is null) then
  /* o campo Job permanece o mesmo */
else
  /* o campo Job foi modificado */
```

A função COALESCE

O Firebird 1.5 tem uma função que pode converter NULL para qualquer outra coisa. Isso permite fazer uma conversão imediata e utilizá-la para processamento posterior, sem a necessidade de construções do tipo *if (MyExpression is null) then*. O nome da função é COALESCE e a chamamos com o seguinte código:

```
COALESCE(Expr1, Expr2, Expr3, ...)
```

COALESE retorna a primeira expressão não NULL em uma lista de argumentos. Caso todas as expressões forem NULL, retornará também NULL. Vejamos como utilizaríamos COALESCE para construir o nome completo de uma pessoa a partir do primeiro, do meio e do último nome, assumindo que alguns campos de nome do meio possam ser NULL:

```
select FirstName
|| coalesce(' ' || MiddleName, '')
|| ' ' || Lastname
from Persons
```

Ou, criamos um nome "tão informal quanto possível" a partir de uma tabela que também inclui apelidos e presumindo que tanto o apelido como o nome podem ser NULL:

```
select coalesce (Nickname, FirstName, 'Mr./Mrs.')
|| ' ' || Lastname
from OtherPersons
```

O COALESCE somente ajudará em situações onde o NULL pode ser tratado do mesmo modo que algum valor permitido para o tipo de dado. Se o NULL precisar de tratamento especial, como no exemplo "direito de votar" utilizado anteriormente, nossa única opção, no final das contas, será apelar para a expressão *if (MyExpression is null) then*.

Firebird 1.0: as funções *NVL

No Firebird 1.0, não existe a função COALESCE, porém, podemos utilizar quatro UDFs que fornecem uma boa parte da sua funcionalidade. Essas UDFs residem na biblioteca *fbudf* e são:

- iNVL: para argumentos de número inteiro;
- i64NVL: para argumentos *bigint*;
- dNVL: para argumentos de precisão dupla;
- sNVL: para seqüências.

As funções utilizam precisamente dois argumentos. Assim como com a função COALESCE, retornam o primeiro argumento caso esse seja não NULL, caso contrário retornam o segundo.

Observem que a biblioteca *fbudf* é do Firebird 1.0 e portanto, as funções estão disponíveis somente para o Windows.

Conclusão

Resumindo: NULL significa *desconhecido*. Caso o NULL figurar em uma expressão, normalmente a expressão inteira se tornará NULL. Em funções agregadas somente os campos não NULL estão envolvidos no cálculo. Exceção do COUNT (*).

As UDFs às vezes convertem NULL <> not NULL de uma maneira aparentemente aleatória. Se a expressão de teste de uma declaração *if* for NULL, então o bloco *if* será pulado e o bloco *else* será executado (caso possua um). Para descobrir se A é NULL, utilizar *A is (not) null*.

O COALESCE e as funções *NVL podem converter o NULL para um valor. A atribuição de NULL é feita como a atribuição de valores: como "A = NULL" ou com uma lista de inserção.

Lembre-se, que mostramos aqui, como o NULL funciona no Firebird, podendo haver diferenças (sutis) em relação a outros bancos de dados.

Referências

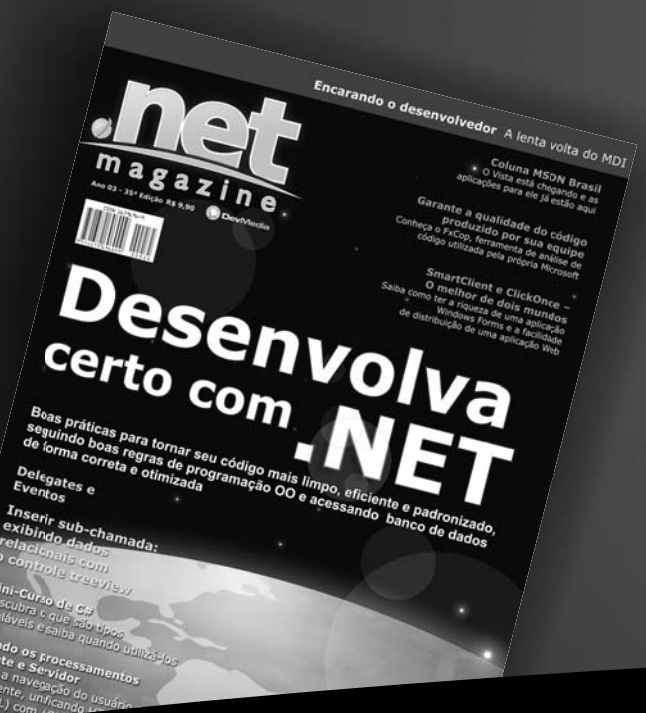
InterBase 6.0 Developer's Guide

www.ibphoenix.com/downloads/60DevGuide.zip

Using Firebird e Firebird Reference Guide (livro)

www.ibphoenix.com

A MAIOR HIDRELÉTRICA DO MUNDO. FEITA POR BRASILEIROS.



Após três anos de publicação decidimos encerrar na edição 35 o uso da marca MSDN Magazine no Brasil. A **MSDN Magazine** agora se chama **.NET Magazine**, uma publicação que tem o orgulho de ser **100% brasileira!** Com a mudança a revista ganhou mais qualidade, mais conteúdo, **mais nacionalidade.**

.NET Magazine
A revista do desenvolvedor que tem orgulho de ser brasileiro.

.net magazine

Já nas bancas!